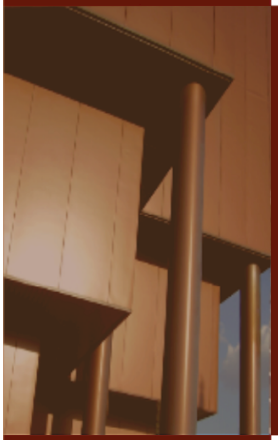
Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

# Desarrollo de entorno de realidad virtual con seguimiento ocular a través de tecnología de infrarrojos para terapia EMDR

Autor: Adán Montero Torres
Tutor: María del Mar Elena Pérez



**Departamento de Ingeniería Electrónica**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

# Desarrollo de entorno de realidad virtual con seguimiento ocular a través de tecnología de infrarrojos para terapia EMDR

Autor:
Adán Montero Torres

Tutor:
María del Mar Elena Pérez
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Grado: Desarrollo de entorno de realidad virtual con seguimiento ocular a través de tecnología de infrarrojos para terapia EMDR

Autor: Adán Montero Torres
Tutor: María del Mar Elena Pérez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

*A mi familia*
*A mis maestros*

# Agradecimientos

A lo largo de todos estos años en la carrera nunca pensé que llegaría este día. Nunca pensé que llegaría el día en el que tuviera que escribir unas palabras para agradecer a toda esas personas que han estado conmigo tanto en los buenos como en los malos momentos, apoyándome. Es por ello, que quiero dedicarles unas palabras a todas esas personas.

Para empezar, nada de esto habría sido posible sin el apoyo incondicional de mis padres. Todo lo que pueda decir de ellos es poco, puesto que siempre han confiando en mí, sabiendo que, tarde o temprano yo iba a sacar esto adelante como siempre había hecho. Por eso, y porque gracias a vuestro esfuerzo de cada día, he hecho posible este sueño de poder dedicar mi vida a lo que realmente me apasiona y es algo que nunca os podré agradecer lo suficiente. Os quiero mucho.

Por supuesto, no me puedo olvidar de mi tía Raquel y de mi abuela. Ellas lo son todo para mí, siempre han estado ahí cuando las he necesitado, haciéndome ver que valgo más de lo que realmente pienso, enseñándome que tirar la toalla no era una opción y que por muy difícil o cuesta arriba que se pusiera todo, no había nada imposible que yo no pudiera hacer. Por todo eso solo puedo deciros lo mucho que os quiero, puesto que diga lo que diga, es poco para expresar lo mucho que os agradezco todo lo que habéis hecho por mí.

Luego están mis tías Isabel y Antonia, sin las cuales no habría podido llegar a donde he llegado sin su apoyo y sin su humor, que hasta en los días más negros, me conseguían sacar una sonrisa, por todo ello, os doy las gracias de corazón.

.

Esto tampoco sería unos agradecimientos si no le dedico unas palabras a mis amigos, tanto a los que siempre han estado ahí, como a los que alguna vez estuvieron, gracias por cada momento compartido conmigo, ya que, sin ellos, no podría estar escribiendo estas palabras.

Y cómo olvidarme de ti, tú, mi punto de apoyo, la que me saca una sonrisa cada día, la que ha tirado de mí cuando nadie más podía, la que ha hecho que aprenda a valorarme y me ha hecho ver el porqué luchar incluso cuando todo está en tu contra. Nunca tendré suficientes palabras para agradecerte todo lo que has hecho por mí y el como has hecho de mí una mejor persona, haciendo de esta etapa de mi vida, algo increíble.

*Adán Montero Torres*

*Sevilla, 2019*

# Resumen

Un trauma o conflicto, es algo que te puede marcar toda la vida, por suerte, hoy día hay cientos de terapias distintas para hacer que supere dichos problemas. Además de eso, vivimos en un mundo cambiante, revolucionado por la tecnología. Un mundo que nos ofrece una gran variedad de herramientas que ponen al alcance de nuestra mano todo un sinfín de información.

Si cogemos estos dos conceptos, el de la terapia por un lado y el de la tecnología por otro, podemos fusionarlos en uno solo para que, con los conocimientos sobre traumas y la tecnología actual pueda llegarse a un punto mas allá en el mundo de la psicología.

Es por esto, que en el presente proyecto se ha desarrollado un sistema mixto con una aplicación que permite al especialista hacer las terapias basadas en la técnica EMDR de una forma mucho más eficiente de lo que ya realmente es. De tal manera que el paciente podrá estar mucho más cómodo al sentir que la terapia no es algo totalmente psicológico, sino que se hace algo más ameno y fácil para esa persona que tiene el problema y que va a enfrentase a ella.

# Índice

# NOTACIÓN

| | |
|---|---|
| EMDR | Eye Movement Desensitization and Reprocessing |
| MPU | Multiple Process Unit |
| IMU | Inertial Measurment Units |
| MEMS | Micro-Electro-Mechanical Systems |
| GITT | Grado en Ingeniería de las Tecnologías de Telecomunicación |
| KB | Kilo Bytes |
| GB | Giga Bytes |
| V | Voltios |

# ÍNDICE DE FIGURAS

# 1 INTRODUCCIÓN

*"El 20% de los esfuerzos generan el 80% de los resultados"*
*Vilfredo Pareto*

En este primer capítulo se va a proceder a dar una breve introducción con el fin de poner en situación al lector de la presente memoria y así abrirle camino en la comprensión de este proyecto además de hacerle conocedor del motivo que impulsó a llevarlo a cabo.

## 1.1 Contexto

Hoy en día, el mundo ha sido sacudido por la tecnología hasta llegar a convertirla en una herramienta indispensable en nuestro día a día, de tal manera que en muchas ocasiones llegamos a depender de ella para poder realizar nuestro trabajo y/o nuestras tareas diarias.

Si nos paramos a pensar, en los países desarrollados, el realizar dichas tareas o trabajos, hacen que el nivel de estrés de una persona puede llegar a límites insospechados. Y no solo eso, existen múltiples factores por los que una persona puede llegar a tener un trastorno de estrés. Es por ello que se buscan formas de eliminar dicho estrés, para obtener una mejor calidad tanto laboral como social.

Estas necesidades dan lugar a la aparición de una abrumadora demanda de nuevas metodologías para detectar dichos problemas a través de nuevos dispositivos tecnológicos y/o aplicaciones.

Es por esto, que la tecnología en general, permite que avancemos hasta el punto de que podamos de conseguir detectar y solucionar un problema de forma cada vez, más precisa.

## 1.2 Presentación del problema

Centrándonos en el ámbito que nos ocupa, el de la psicología, son muchas las formas que hay de tratar un trauma, es por esto, que nace la necesidad de crear una tecnología y/o aplicación, donde el propio psicólogo pueda englobar una única forma de tratarlos, sea capaz de identificarlos a través de la respuesta del paciente ante un entorno inmersivo donde pueda revivir dicho conflicto, y con la ayuda de la técnica del EMDR [2,8], concepto que será desarrollado y explicado en secciones posteriores del presente documento, poder tratarlos.

De esta manera, la mayoría de personas que sufren un trauma, o personas que tienen un comportamiento impropio a su forma de ser debido a que sufren un trauma y no lo saben, podrían tratarse con esta nueva tecnología que tiene como base la técnica del EMDR.

En definitiva, el objetivo de este proyecto es desarrollar un dispositivo y/o aplicación única y personalizable para cada individuo, con entornos inmersivos diferentes en función de sus problemas, mientras el psicólogo hará un seguimiento ocular del paciente.

## 1.3 Antecedentes

El presente proyecto es la creación de una mejora basada en el descubrimiento de la Dra. Francine Saphiro en 1987, la cual, observó casualmente, como los movimientos oculares que respondían al seguimiento de sus dedos, disminuían los niveles ansiosos al pensar en un evento estresante específico. Dicho proceso, pasó a denominarse más tarde como EMDR (“Eye Movement Desensitization and Reprocessing” o lo que es lo mismo “Desensibilización y reprocesamiento mediante movimientos oculares”).

Con el paso de los años, el EMDR pasó a convertirse en una terapia bastante eficiente en la que el paciente, siempre guiado por el terapeuta, reestructure la información almacenada de manera disfuncional que se encuentra en su memoria provocando un cambio primero a nivel emocional para luego dar paso al cambio cognitivo, esto conlleva a una nueva percepción del incidente traumático sin las emociones previas que desregulan afectivamente a la persona y causan malestar emocional o síntomas asociados a la esfera ansiosa o depresiva

Hoy en día, la terapia EMDR está avalada por la Organización Mundial de la Salud y las Guías Clínicas Internacionales para el tratamiento del trauma. Se basa en la comprensión del efecto de las experiencias vitales adversas y traumáticas sobre la patología y en el procesamiento de dichas experiencias a través de procedimientos estructurados que incluyen movimientos oculares u otras formas de estimulación bilateral. Su aplicación se ha extendido a un amplio rango de problemas clínicos.

**Otras referencias**

Finalmente, cabe destacar, que este proyecto es la continuación del trabajo final de la asignatura "Proyectos de Sistemas Electrónicos de 4º de GITT", en donde se tomó un primer contacto en qué se basa la terapia así como una prueba inicial de conceptos. Todo ello profundizado en el Anexo C.

Es por esto, que en el presente proyecto se ha profundizado más aún y se ha intentado dar un paso más allá, intentando buscar una solución lo más real y funcional posible.

## 1.4 Descripción de la solución

En este apartado se detallarán los objetivos impuestos en la realización del proyecto, así como la funcionalidad y arquitectura que finalmente presenta la solución dada.

### 1.4.1 Objetivos

El objetivo principal que se ha perseguido ha sido el poder ofrecer al terapeuta o psicólogo, una herramienta con la que poder hacer una terapia de la forma más eficiente posible, siendo capaz de crear entornos inmersivos, a través de imágenes y estímulos bilaterales, donde el paciente pueda recrear perfectamente sus conflictos y puedan ser tratados, mediante la terapia de EMDR.

### 1.4.2 Características y funcionalidades

Para poder lograr dicho objetivo, se desarrollará una tecnología basada en realidad virtual y 3D, que tendrá como funcionalidad el poder generar los siguientes estímulos:

- Visuales:
  - Consistirá en una serie de imágenes y/o videos para cada paciente, con la finalidad de crear un entorno lo más inmersivo y personalizado posible a la hora de enfrentar el trauma.

- Sonoros:

  Dos tipos:

  - Por un lado, el sonido 3D, el cual, se sincronizará con las escenas que se estén viendo en los estímulos visuales, para dar más sensación de realismo.
  - Por otro lado, los estímulos bilaterales, que crearan sensación de profundidad e inmersión.

- Además, se clasificarán la mayoría de tipos de traumas a tratar durante esta terapia:
  - Abuso sexual.
  - Experiencias humillantes.
  - Accidentes.
  - Lesiones.
  - Violencia o presenciar episodios de violencia (sobre todo el en ámbito familiar).
  - Estrés implacable
  - Desastres naturales.
  - Defunción de un familiar o amigo cercano.
  - Ruptura de una relación significativa.
  - Experiencias profundamente decepcionantes.

### 1.4.3 Diseño

La solución propuesta para abarcar todo lo dicho anteriormente será el diseño un sistema compuesto por: unas gafas de realidad virtual y altavoces para audio 3D/Bilateral.

Todo eso estará bajo la supervisión de un profesional que estará en la sala con el paciente, como podemos apreciar en el siguiente esquema:

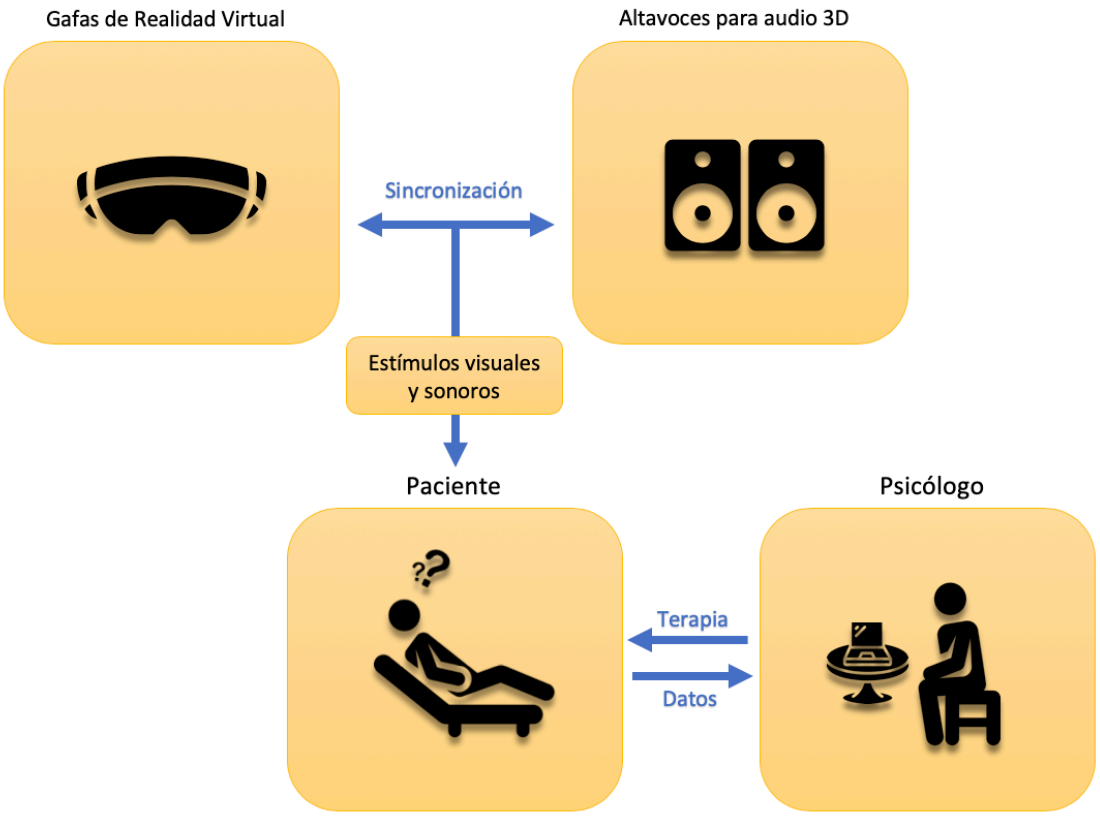



*Ilustración 1: Esquema de funcionamiento del proyecto*

### 1.4.4 Gafas de Realidad Virtual

Para el estímulo visual, se precisará de unas gafas de realidad virtual convencionales, que sea capaz de crear un entorno con imágenes y/o videos personalizados, pero con una particularidad, estas han de tener un sensor de seguimiento ocular o Eye Tracking, para poder aplicar correctamente la técnica de EMDR, además, estas estarán sincronizadas con el audio 3D/Bilateral.

### 1.4.5 Altavoces para audio 3D

Como hemos dicho anteriormente, otro de los estímulos es el sonoro, que podía ser 3D o Bilateral, para ello necesitaremos unos altavoces capaces de reproducir dicho sonido con la suficiente eficiencia para poder dar un entorno totalmente inmersivo, además, audio irá sincronizado con las imágenes de las gafas.

### 1.4.6 Paciente y Psicólogo

El psicólogo le hará un estudio previo al paciente para saber como empezar la terapia y qué temas debe abordar. Mientras tiene lugar la terapia, el psicólogo le irá haciendo preguntas al paciente a la vez que ve en su ordenador como responde a través del sensor ocular. Por su parte, el paciente, se encontrará inmerso en la terapia contestando a las preguntas y dejándose llevar para resolver su conflicto.

## 1.5 Estructura de la memoria

La presente memoria está dividida en una serie de capítulos numerados en los que se irá profundizando en el funcionamiento y estructura de la aplicación que comprenden este proyecto. A continuación, se da una breve descripción de cada uno de esos capítulos para poner en situación al lector.

*Capítulo 1: Introducción*

Sitúa al lector en el ámbito que concierne a este proyecto y da una breve descripción de la funcionalidad y arquitectura de éste con el fin de ir profundizando en dichos aspectos en próximos capítulos.

*Capítulo 2: Laboratorio*

Se hace un repaso por todos los diseños y esquemas finales utilizados en el proyecto.

*Capítulo 3: Tecnologías y Herramientas utilizadas*

Describe las tecnologías y herramientas utilizados para llevar a cabo con éxito el proyecto y el porqué de la elección de éstos.

*Capítulo 4: Software y funcionalidad*

Se explican todos los códigos utilizados, la forma de programar y el porqué. Además, se explicará la funcionalidad final del proyecto.

*Capítulo 5: Problemas encontrados*

Se detalla cada uno de los problemas encontrados durante la realización el proyecto y el cómo se han solventado estos.

*Capítulo 6: Líneas futuras*

Se mencionan posibles futuras mejoras o cambios de tecnologías para mejorar la funcionalidad y eficiencia.

*Capítulo 7: Planificación y presupuesto*

Se detalla el cómo se ha planificado el proyecto, cuánto tiempo se ha estimado para cada parte y en desglose del presupuesto total de este.

*Capítulo 8: Conclusiones*

Se incluye una conclusión tras la finalización del proyecto analizando lo realizado y haciendo una valoración personal de ello.

*Capítulo 9: Referencias*

Se incluye una conclusión tras la finalización del proyecto analizando lo realizado y haciendo una valoración personal de ello.

*Capítulo 10: Anexo A: Códigos del proyecto*

*Capítulo 11: Anexo B: Instalación de Raspbian.*

*Capítulo 12: Anexo C: Requisitos especiales..*

# 2 LABORATORIO

En este capítulo se va a proceder a presentar el diseño escogido para una de las partes del proyecto y que han permitido el funcionamiento de este. Además, se detallará cada parte del hardware escogido y se enfatizará en la finalidad con la que han sido usadas.

## 2.1 Gafas de Realidad Virtual

El diseño hardware de las gafas consta de tres elementos: Arduino™ genuino UNO, Giroscopio y Acelerómetro MPU-6050 y Raspberry™ Pi 3 B, interconectados entre sí como vemos en la siguiente imagen:

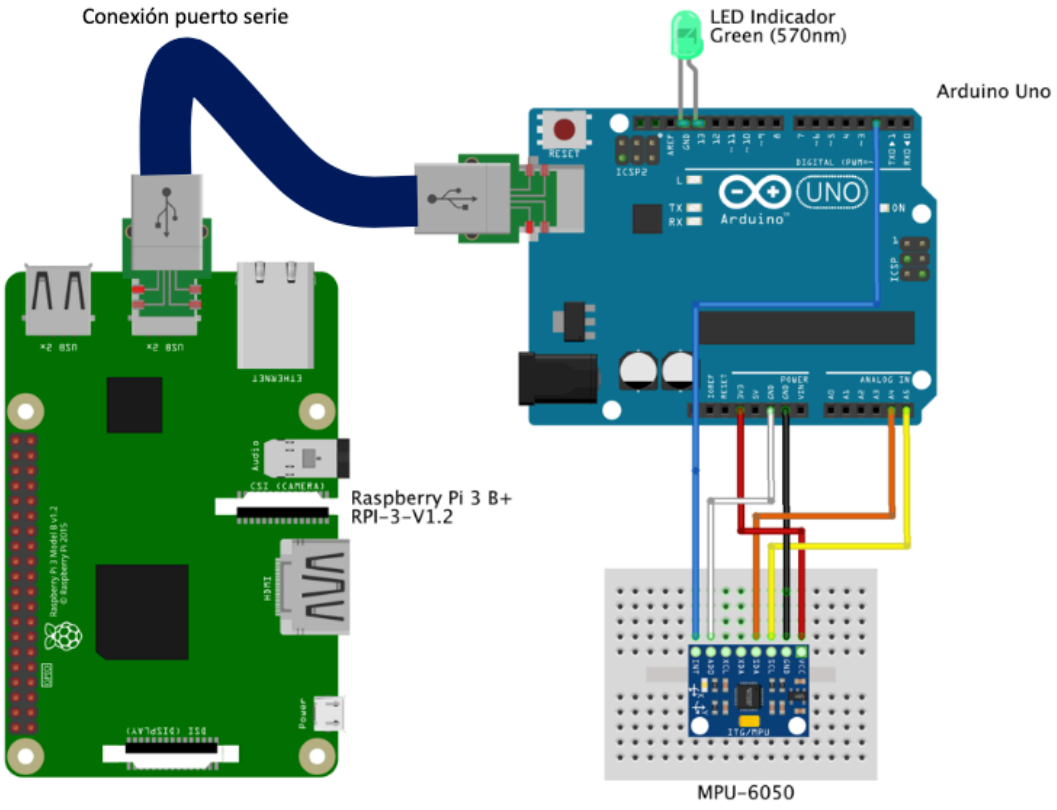



*Ilustración 2: Esquema general hardware gafas VR*

A continuación, se pasará a detallar las funcionalidades de cada una de las partes de la imagen anterior, explicando para qué sirve cada una y la función que cumplen.

### 2.1.1 MPU-6050

El MPU-6050 [7] es una unidad de medición inercial o IMU (Inertial Measurment Units) de 6 grados de libertad pues combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes.

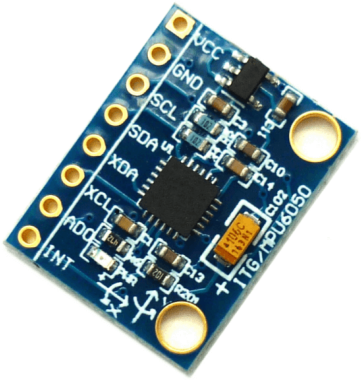



*Ilustración 3: MPU-6050*

EL MPU-6050 será el encargado de dar la posición y velocidad, es decir, el movimiento necesario para poder hacer un entorno de realidad virtual. A continuación, se detallará unas nociones básicas sobre ambas partes:

#### 2.1.1.1 Aceleración y acelerómetros

La aceleración es la variación de la velocidad por unidad de tiempo es decir razón de cambio en la velocidad respecto al tiempo:

$$a=dV/dt$$

Así mismo la segunda ley de Newton indica que en un cuerpo con masa constante, la aceleración del cuerpo es proporcional a la fuerza que actúa sobre él mismo:

$$a=F/m$$

Este segundo concepto es utilizado por los acelerómetros para medir la aceleración. Los acelerómetros internamente tienen un MEMS (MicroElectroMechanical Systems) que de forma similar a un sistema masa resorte permite medir la aceleración.

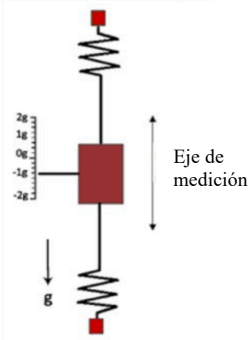
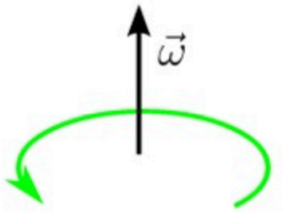



*Ilustración 4: Concepto medida aceleración MPU-6050*

#### 2.1.1.2 Velocidad angular y giroscopio

La velocidad angular es la tasa de cambio del desplazamiento angular por unidad de tiempo, es decir que tan rápido gira un cuerpo alrededor de su eje:




$$\omega = \frac{d\theta}{dt}$$

*Ilustración 5: Concepto medida velocidad angular MPU-6050*

Los giroscopios utilizan un MEMS (MicroElectroMechanical Systems) para medir la velocidad angular usando el efecto Coriolis.

Con un giroscopio podemos medir la velocidad angular, y si se integra la velocidad angular con respecto al tiempo se obtiene el desplazamiento angular (posición angular si se sabe dónde se inició el giro, algo necesario para nuestro proyecto).

#### 2.1.1.3 Módulo Acelerómetro y Giroscopio MPU-6050

El módulo Acelerómetro MPU tiene un giroscopio de tres ejes con el que podemos medir velocidad angular y un acelerómetro también de 3 ejes con el que medimos los componentes X, Y, Z de la aceleración.

La dirección de los ejes está indicada en el módulo, el cual, hay que tener en cuenta para no equivocarnos en el signo de las aceleraciones.

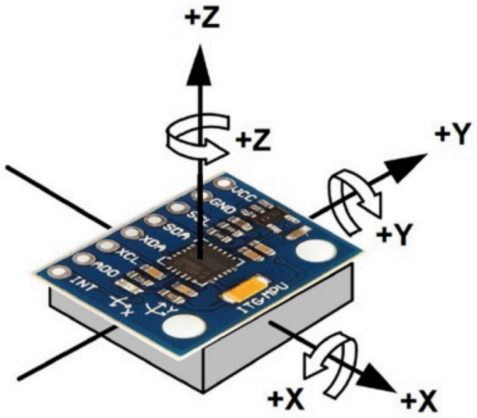



*Ilustración 6: Disposición coordenadas MPU-6050*

#### 2.1.1.4 Datos técnicos del MPU-6050

A continuación, se detallará una serie de datos técnicos, de los cuales, se profundizará más a fondo en capítulos posteriores:

- La comunicación del módulo es por I2C, esto le permite trabajar con la mayoría de microcontroladores.
- Los pines SCL y SDA tienen una resistencia pull-up en placa para una conexión directa al microcontrolador o Arduino.
- El pin ADDR internamente en el módulo tiene una resistencia a GND, por lo que, si no se conecta, la dirección por defecto será 0x68.
- El módulo tiene un regulador de voltaje en placa de 3.3V, el cual se puede alimentar con los 5V del Arduino.

#### 2.1.1.5 Detalles del conexionado

Por último, el dispositivo descrito se conectará al Arduino tal y como se muestra en la imagen siguiente, donde vemos como se conecta alguno de los pines antes mencionados.

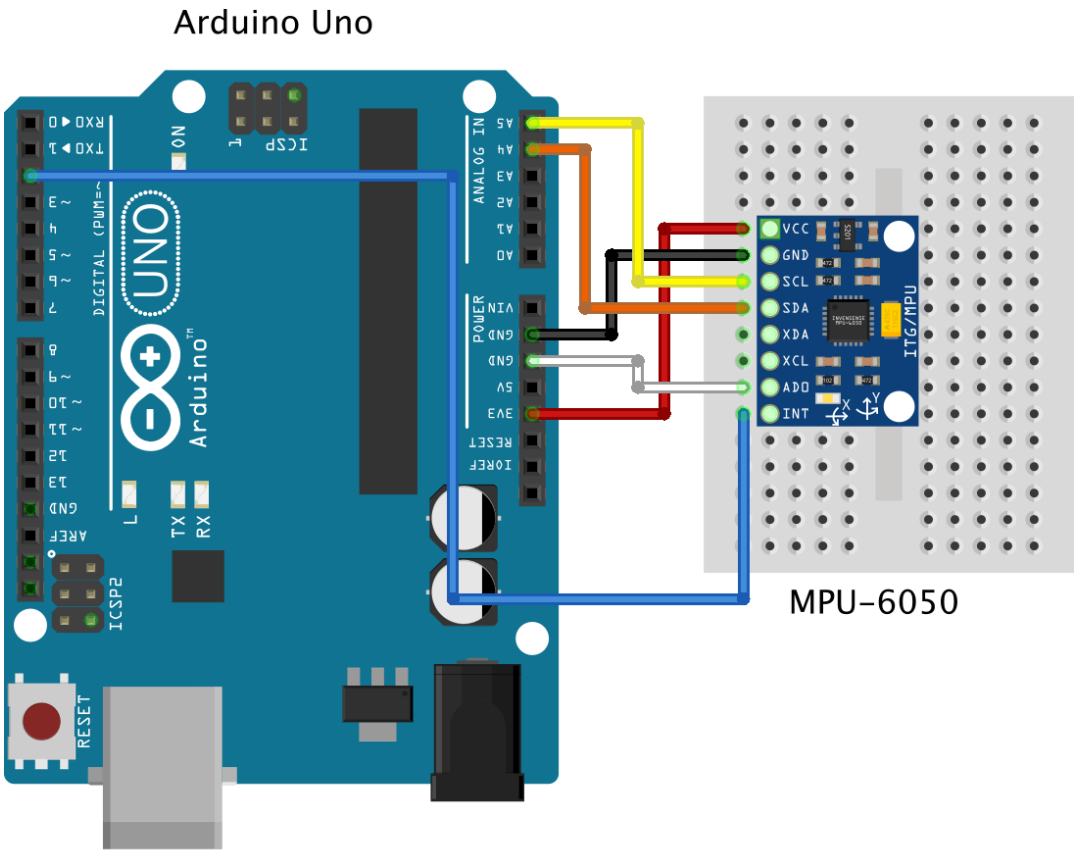



*Ilustración 7: Conexionado del MPU-6050*

### 2.1.2 Arduino UNO

Arduino [5,7] es una plataforma de hardware y software libre, en concreto, la que usaremos para este proyecto, **Arduino UNO** es una placa electrónica de las muchas que tiene Arduino y con la que es muy fácil introducirse en el mundo de la programación electrónica.


*Ilustración 8: Arduino UNO*

En esta parte del proyecto, el Arduino será el nexo de conexión principal, como podemos ver en la *Ilustración 2*, es decir, hará la función de transmitir los datos proporcionados por el MPU-6050 (cómo hemos visto anteriormente), a través de su puerto serie hacia la Raspberry pi 3 B+ [3,11,12], la cual, veremos su función más adelante.

#### 2.1.2.1 Datos técnicos Arduino UNO

Entre sus características, nos encontramos que Arduino UNO puede ser alimentada de varias formas, con un cable USB conectado al ordenador, ya que posee un conector USB tipo-B hembra o bien, con una fuente externa a través del zócalo para el Jack de 2,1mm que posee. Además de esto, posee las siguientes partes y características:

- Microcontrolador ATmega328.
- Voltaje de 5 voltios.
- Voltaje de entrada recomendado de 7-12 voltios.
- 14 pines de Entrada/Salida Digital (pines 0-13), de las cuales 6 son para PWM.
- 6 pines de Entradas Analógicas (pines A0-A5).
- Memoria Flash de 32KB

- SRAM de 2KB
- EEPROM de 1KB
- Velocidad de Reloj de 16 MHz
- Botón de Reset
- Botón de ICSP

### 2.1.2.2 Detalles del conexionado

En la siguiente imagen, podemos ver la disposición de alguno de los elementos antes mencionados, de los cuales nos vamos a centrar en los pines más destacados.

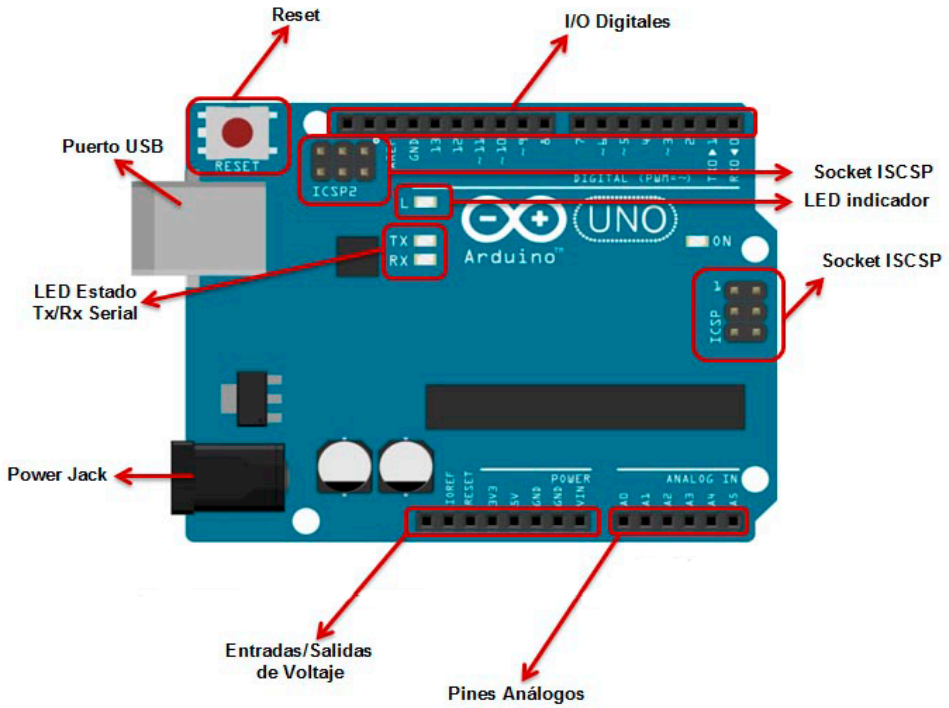



*Ilustración 9: Conexionado Arduino UNO*

**Pin VIN:**

Este pin se puede usar de varias formas, si tenemos una fuente de alimentación conectada mediante un adaptador, lo que podemos hacer mediante este pin es obtener la alimentación para conectar otro dispositivo, pero tenemos que tener en cuenta que la placa no regulara la tensión y obtendremos la misma tensión que tenga el adaptador. Por otro lado, si tenemos conectado el USB, la tensión será regulada a 5v.

**Pin GND:**

El pin GND referencia tierra.

**Pin 5V:**

Este pin tiene varias funciones, podemos alimentar la placa mediante este pin, siempre que tengamos la fuente externa regulada a 5v. Por otro lado, si tenemos la placa alimentada tanto por el Jack como por USB, se puede alimentar otro componente con una tensión regulada de 5v.

**Pin 3.3V:**

Por este pin sacamos una tensión de 3.3v que es alimentada mediante el conector Jack o el USB. Los 3.3v se utilizan para alimentar dispositivos que requieren una tensión baja.

**Pines de Entradas Analógicas:**

La placa de Arduino cuenta con 6 pines de entradas analógicas, que van desde el pin A0 al A5, de los cuales proporcionan 10bits, llamados bits de resolución. La tensión que miden va de 0 a 5v.

**Pin IOREF:**

El pin IOREF es una copia del pin VIN y se utiliza para indicar a los demás dispositivos conectador a la placa que las tensiones de los pines de entrada y salida son 5v.

**Pin RESET:**

Este pin tiene el mismo funcionamiento que el botón RESET, se utiliza para reiniciar el microcontrolador.

**Pines de Entrada/Salida Digital:**

Las entradas y salidas digitales son 14 y van desde el pin 0 al 13 y ofrecen una tensión de 5v.

**Pines A5 SCL y A4 SDA:**

Se pueden utilizar para conectar dispositivos que lleven a cabo comunicaciones mediante la librería Wire (Librería usada para comunicación I2C, imprescindible para nuestro proyecto).

**Pin AREF:**

Ofrece un voltaje de referencia para las entradas analógicas.

**Pines 1 TX y 0 RX:**

Estos pines se iluminan cuando reciben y/o transmiten datos en serie.

### 2.1.3 Raspberry Pi 3B+

Raspberry Pi, es un es un ordenador de tamaño de tarjeta de crédito que se conecta a su televisor y un teclado. Es una placa que soporta varios componentes necesarios en un ordenador común. Es un pequeño ordenador capaz, que puede ser utilizado por muchas de las cosas que su PC de escritorio hace, como hojas de cálculo, procesadores de texto y juegos. También reproduce vídeo de alta definición.

*Ilustración 10: Raspberry Pi 3B+*

La Raspberry es la etapa final, será la encargada de toda la parte visual, es decir, la encargada de procesar el programa capaz de crear el entorno de realidad virtual con las imágenes y/o videos necesarios. Para ello, se conectará a Arduino a través de uno de sus puertos USB para recibir los datos por comunicación serie, como podemos ver en la *Ilustración 2*. Posteriormente, irá conectado a una pantalla o monitor a través de su puerto HDMI.

#### 2.1.3.1 Datos técnicos Raspberry Pi 3B+

En términos de potencia, esta Raspberry tiene una potencia bastante aceptable, tal es así, que entre los usos más comunes de este micro-ordenador destacan el usarse como una retro-consola, hacer las funciones de NAS o incluso de servidor web casero.

Entonces, para poder soportar lo dicho anteriormente debe tener unas características y prestaciones como las siguientes:

- Procesador Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC.
- Frecuencia de reloj de 1.4 GHz.
- Memoria RAM de 1GB LPDDR2 SDRAM.
- Wi-Fi de 2.4 GHz / 5 GHz con estándar IEEE 802.11.b/g/n/ac.
- Bluetooth 4.2.
- Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps).
- GPIO de 40 pines.
- HDMI.
- 4 puertos USB 2.0.
- Puerto CSI y DSI para conectar una cámara y una pantalla táctil.
- Salida de audio estéreo y vídeo compuesto.
- Micro-SD.
- Power-over-Ethernet (PoE).

### 2.1.3.2 Detalles del conexionado

En la siguiente imagen podemos ver la disposición de algunas de las características antes mencionadas, así como los 40 pines de placa al detalle:

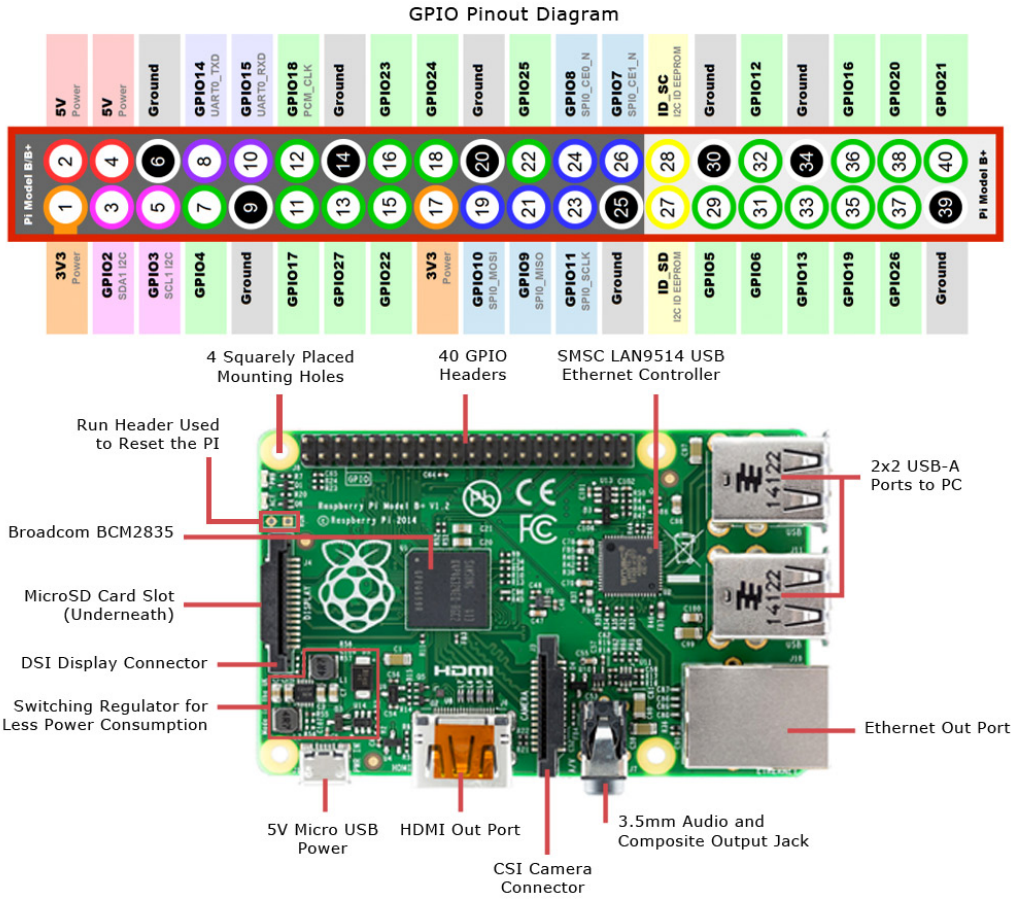



*Ilustración 11: Conexionado Raspberry Pi 3B+*

## 2.2 Sensor de movimiento ocular

En cuanto al hardware que usaremos para hacer el Eye Tracking o Sensor Ocular consistirá de solo dos partes, las cuales son: Arduino Pro Mini y una Webcam con 4 diodos leds infrarrojos conectados entre sí, como podemos ver en el siguiente esquema:

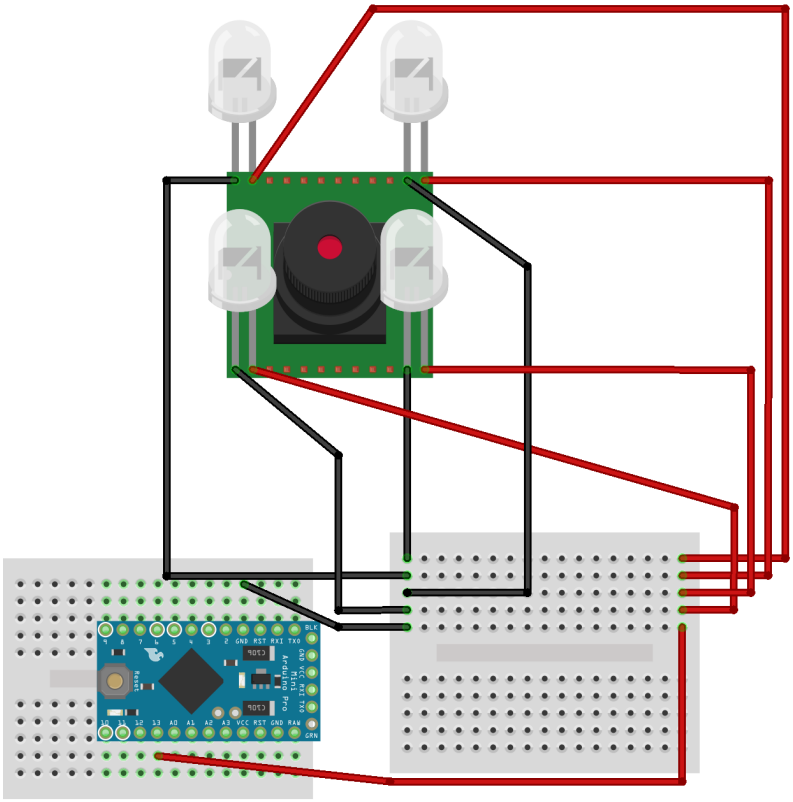


*Ilustración 12: Esquema general hardware Eye Tracking*

Cómo podemos observar es una estructura más simple que la de las gafas, pero tiene algo más de complicación como veremos a continuación viendo la función de cada uno de los componentes.

### 2.2.1 Arduino Pro Mini

Como ya dijimos antes, la marca Arduino tiene muchos tipos de placas y el **Arduino Pro Mini** es otra con un poco meno de potencia que Arduino UNO, pero más pequeño y transportable aún que este anterior, como podemos ver en la siguiente imagen comparativa.

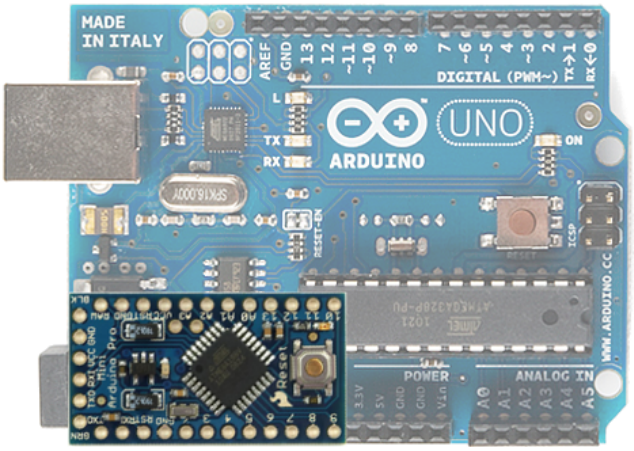



*Ilustración 13: Tamaño Arduino Pro Mini vs Arduino UNO*

Por esto mismo, al ser tan pequeño y manejable, se utilizará este tipo de Arduino para esta parte del proyecto, puesto que el sensor ocular debe de ser algo pequeño, además, de que se si se alimenta con una batería es totalmente independiente y no haría falta que estuviera conectado a un ordenador, aunque en este caso, se optará usar de un conversor USB a Serial (TTL) poder programarlo y alimentarlo a través de él, como podemos ver a continuación:

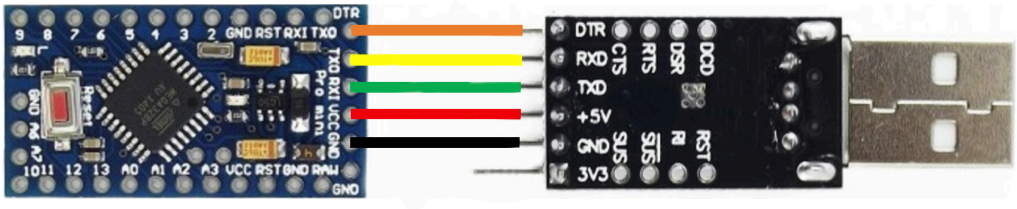



*Ilustración 14: Arduino Pro Mini conectado a conversor USB-Serial*

#### 2.2.1.1 Datos técnicos Arduino Pro Mini

Existen dos versiones de Arduino Pro Mini. Una que funciona a 3.3V/8MHz y otra que funciona a 5V/16MHz, esta última será la que utilizaremos en nuestro proyecto. Además de esto, las características de este Arduino son:

- Microcontrolador ATmega328P.
- Voltaje de 3.3V ó 5V.

- 14 pines de Entrada/Salida Digital (pines 0-13).
- Puerto serie UART.
- Puerto bus SPI e I2C.
- 6 entradas Analógicas (pines A0-A5).
- Memoria Flash de 32KB.
- SRAM de 1KB.
- EEPROM de 1KB.
- Frecuencia de reloj de 8MHz ó 16MHz.
- Botón de RESET.

#### 2.2.1.2 Detalles del conexionado

En la siguiente imagen podemos ver como se distribuyen los elementos antes mencionados con un pequeño esquemático de cada uno de ellos:

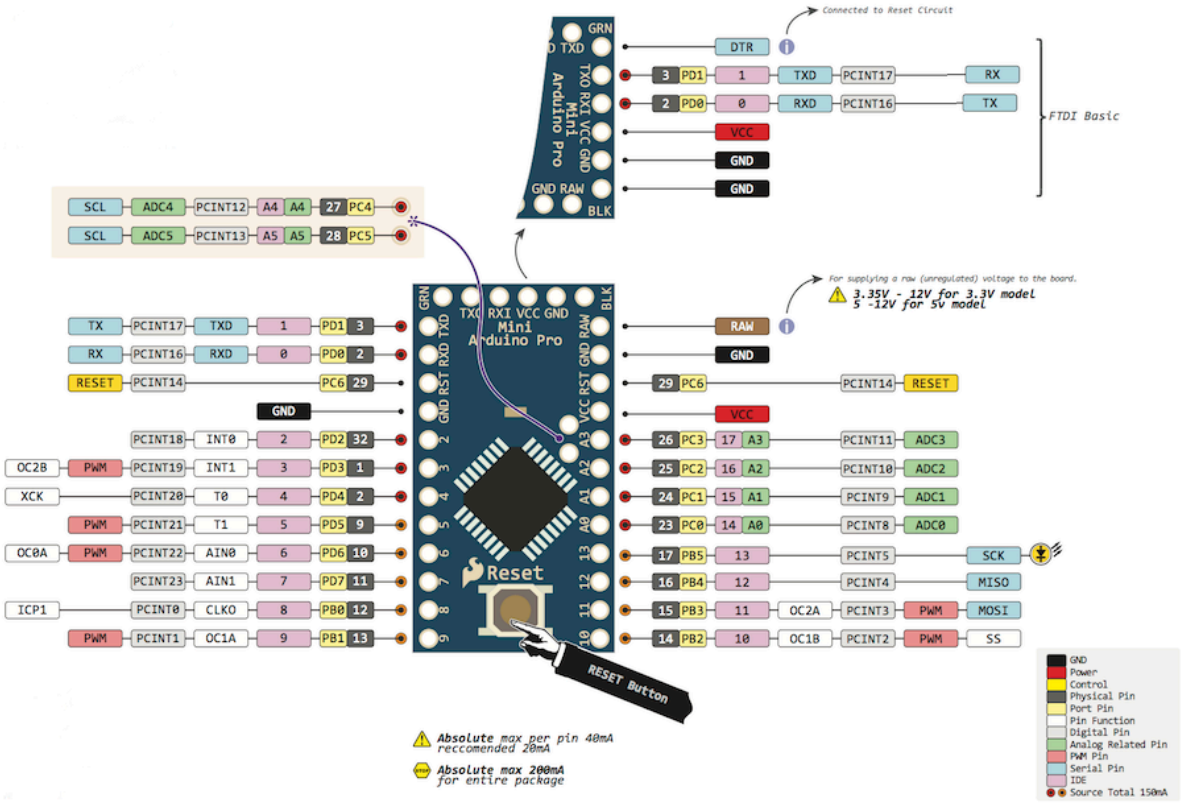



*Ilustración 15: Conexionado Arduino Pro Mini*

### 2.2.2 Webcam con diodos infrarrojos

Para que se pueda llevar a cabo el seguimiento ocular, en nuestro caso, precisaremos de una webcam convencional como la siguiente:


*Ilustración 16: Webcam convencional*

Una vez tengamos la cámara, pasaremos a realizar una serie de transformaciones en ella para que pueda hacer la función de seguimiento ocular:

1. **Despiece webcam**

   Esta es, sin duda, la parte más experimental de todo el proyecto, pues consiste en abrir la cámara web para eliminar el filtro infrarrojo de su lente:

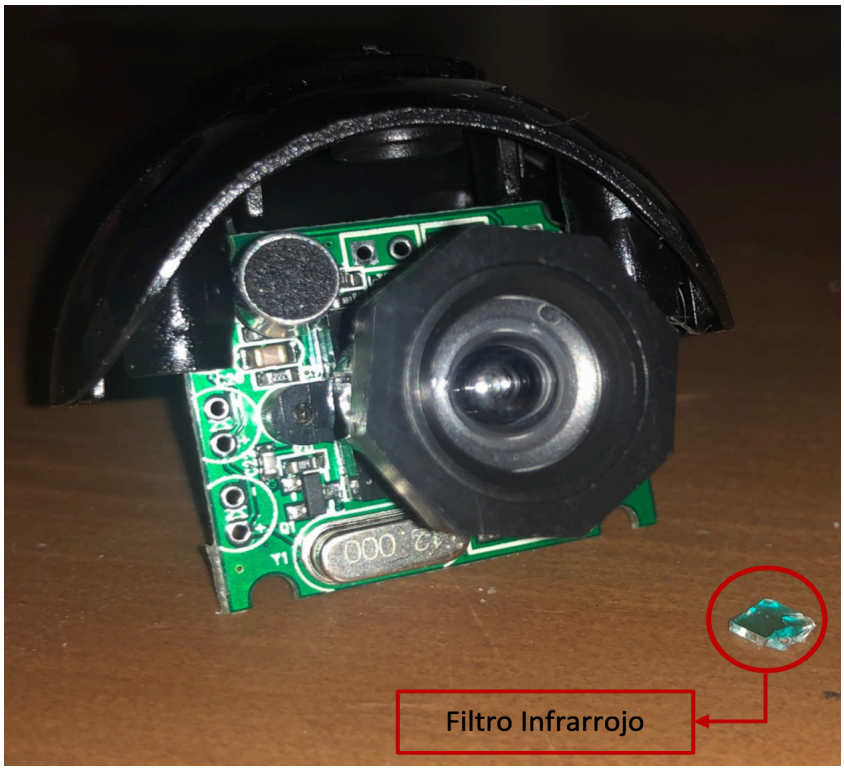



*Ilustración 17: Despiece y eliminación filtro IR*

**2. Acople de leds infrarrojos**

Una vez tenemos la cámara desmontada y sin filtro infrarrojo, tocará colocar los 4 leds infrarrojos alrededor de la lente (como podemos ver en la *Ilustración 12*) para que quede de la siguiente forma:

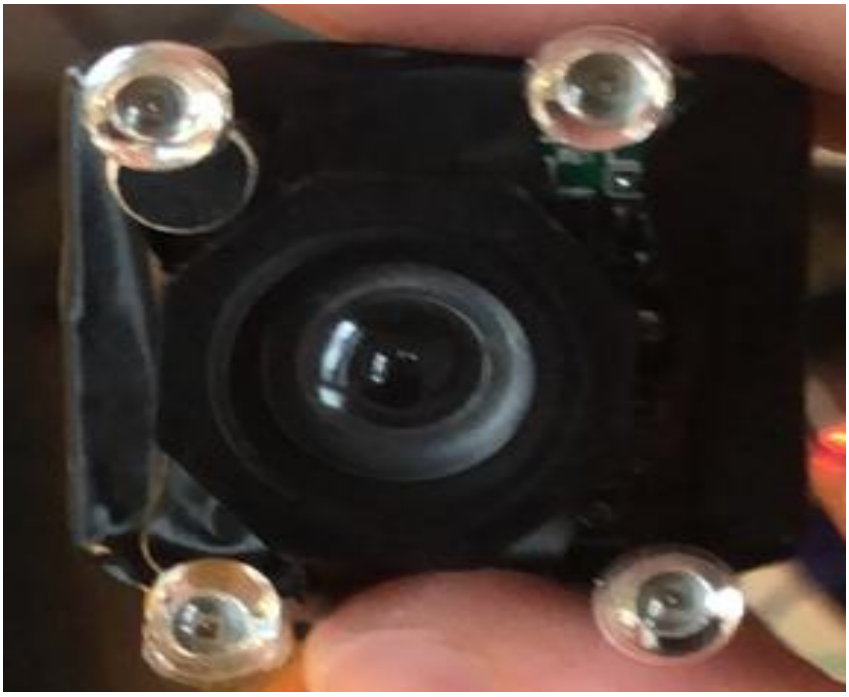


*Ilustración 18: Cámara con leds IR*

Con dichas modificaciones, ya tendríamos la cámara preparada para que pueda funcionar como sensor ocular, el cual, se explicará como funciona, en capítulos posteriores.

# 3 TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS

En este capítulo se va a proceder a presentar las tecnologías utilizadas en el proyecto y las herramientas que han permitido su correcto funcionamiento. Además, se detallará cada una de ellas y se enfatizará en la finalidad con la que han sido usadas.

## 3.1 Tecnologías y herramientas en las gafas VR

### 3.1.1 Arduino

#### 3.1.1.1 Arduino como lenguaje

Arduino (.ino) es el lenguaje de programación para las placas de la familia Arduino, valga la redundancia, que está basado en C++, no un C++ puro, sino que es una adaptación proveniente de una librería de C de alta calidad para usar con GCC (compilador de C y C++). La principal característica de este lenguaje de programación es su sencillez y facilidad de uso.

Por tanto, se puede decir que, Arduino es una forma sencilla de realizar proyectos interactivos para cualquier persona el proceso pasa por descargarnos e instalar el IDE, del cual hablaremos más adelante, buscar un poco por internet y simplemente hacer “corta y pega” del código que nos interese y cargarlo en nuestra placa.

Aunque se hable de que hay un lenguaje propio de programación de Arduino, no es cierto, la programación se hace en C++, pero Arduino ofrece una api o core que facilitan la programación de los pines de entrada y salida y de los puertos de comunicación, así como otras librerías para operaciones específicas. El propio IDE ya incluye estas librerías de forma automática y no es necesario declararlas expresamente. Otra diferencia frente a C++ estándar, es la estructura del programa que ya hemos visto anteriormente.

La estructura básica de un sketch de Arduino es bastante simple y se compone de al menos dos partes principales: setup () y loop (), como vemos a continuación:

```
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

*Ilustración 19: Funciones principales del lenguaje Arduino*

Estas dos partes son obligatorias y encierran bloques que contienen declaraciones, estamentos o instrucciones y podríamos definirlas de la siguiente manera:

- **setup ()** es la parte encargada de recoger la configuración deseada.
- **loop ()** es la que contiene el programa que se ejecuta cíclicamente (de ahí el término loop –bucle-).

Por último, pero no menos importante, cuando compilamos y cargamos el programa en nuestra placa Arduino esto es lo que ocurre:

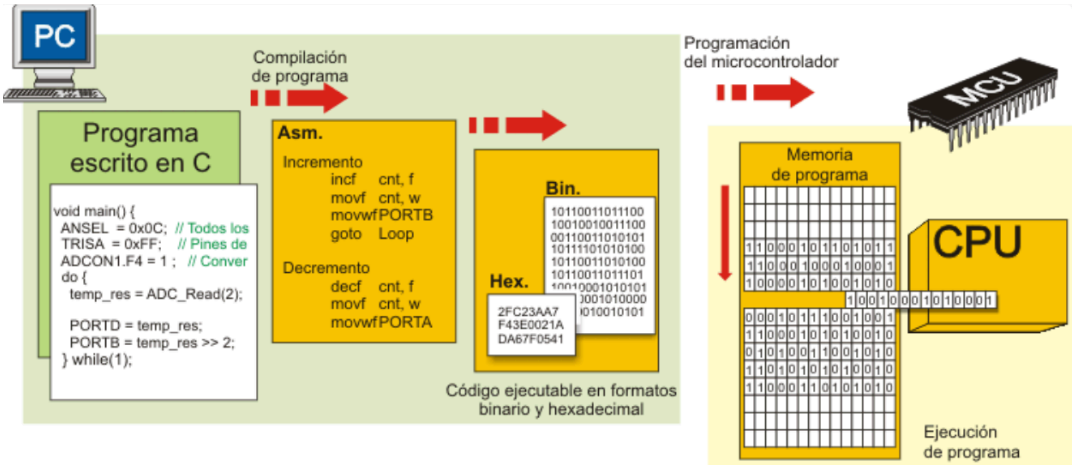



*Ilustración 20: Proceso de carga y ejecución Arduino*

#### 3.1.1.2 Arduino como entorno de desarrollo

El entorno de programación de Arduino se llama IDE (Integrated Development Environment) es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Además, en el caso de Arduino incorpora las herramientas para cargar el programa ya compilado en la memoria flash del hardware a través del puerto serie.

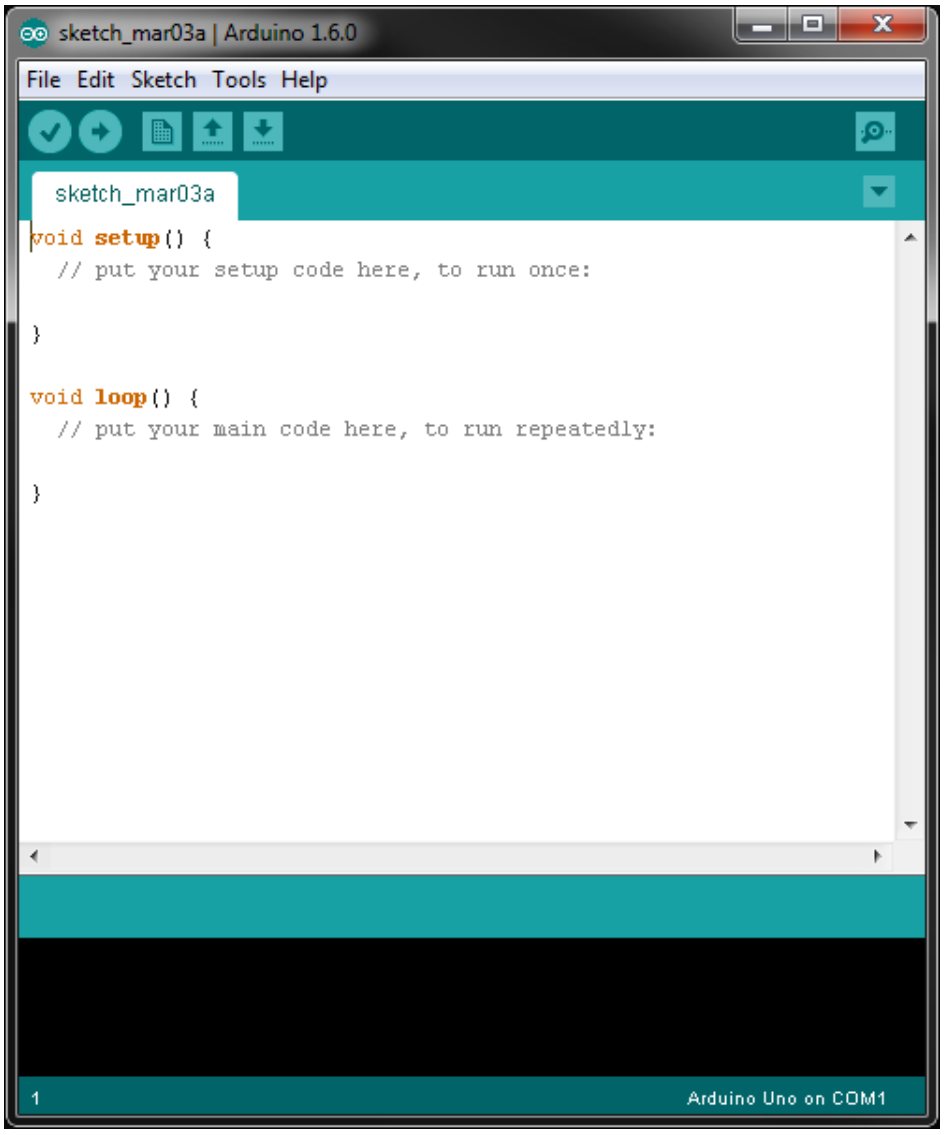



*Ilustración 21: IDE Arduino*

Además, es destacable la gestión de librerías, donde puedes añadir desde el propio programa o descargarlas de terceros, y la gestión de placas, donde tenemos todas las placas de la familia Arduino hasta la fecha.

Por último, los programas de Arduino están compuestos por un solo fichero con extensión "ino", aunque es posible organizarlo en varios ficheros. El fichero principal siempre debe estar en una carpeta con el mismo nombre que el fichero.

### 3.1.2 Processing™

#### 3.1.2.1 Processing como lenguaje

Processing (.pde) [14] es un dialecto de Java que fue diseñado para el desarrollo del arte gráfico, para las animaciones y aplicaciones gráficas de todo tipo. Desarrollado por y para artistas.

Es un software basado en Java, concretamente en la versión 1.4.2 y, por lo tanto, multiplataforma. Tiene la gran ventaja que se puede utilizar con plataformas como Arduino que, además, se trata de un aprendizaje plug and play, es decir, no tenemos que andar configurando e instalando miles de dependencias para ponernos a trabajar.

Es una tecnología que integra entorno de desarrollo, el cual se profundizarán en apartados posteriores, y lenguaje de programación. Al contrario de las alternativas que tenemos para desarrollar para entornos gráficos como **OpenGL**, el cual es bastante complicado y engorroso, Processing nos facilita esta tarea y nos evita la frustración cuando queremos aprender un lenguaje de estas características.

Además, al igual en el lenguaje Arduino, también existen dos funciones principales que deben estar en todo fichero de programación .pde, que son:

- **setup ()** es donde se inicializan las variables de nuestro boceto.
- **draw ()** es la una función equivalente al *loop de* Arduino.

Existen 3 formas de programar este lenguaje:

- De forma básica, tipo Basic o Ensamblador. Sentencia a sentencia, con variables globales y sin nada de complejidad.
- Podemos utilizar la programación procedural o estructurada como en C. Algo más compleja pero mucho más limpia donde crearemos nuestras propias funciones a las que posteriormente llamaremos (Modo en el que programaremos en nuestro proyecto).
- También podemos utilizar la manera más compleja, utilizando toda la potencia de la programación orientada a objetos.

Una vez se elija la forma de programar, y se haga un programa, el proceso por el cual Processing hace un ejecutable, es el siguiente:

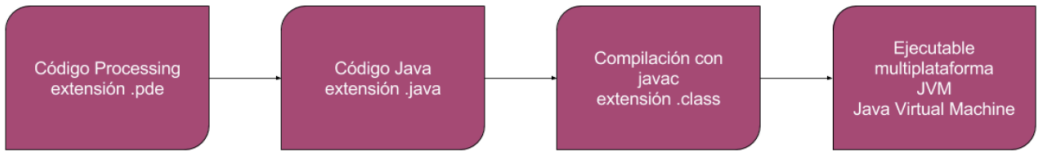



*Ilustración 22: Proceso para generar ejecutable de Processing*

#### 3.1.2.2 Processing como entorno de desarrollo

Processing tiene su propio entorno de desarrollo, la cual, se llama PDE (Processing Development Enviroment) desarrollado en Java.

En la interfaz de Processing, nos fijaremos en 3 elementos principales:

- El área blanca que ocupa la mayor parte de la interfaz: en esta área será donde escribamos nuestras líneas de código.
- El botón de "play" en la esquina superior izquierda de la interfaz: Este botón sirve para indicarle a Processing que compile y ejecute el programa que acabamos de escribir.
- La consola: es el área negra de la parte inferior, donde podremos recibir mensajes del programa que nos ayuden a inspeccionar su ejecución.

Es muy sencillo, intuitivo y fácil debido a que ésta funciona principalmente como un editor de texto, que nos servirá para compilar el código de los programas que escribamos, como podemos ver en la siguiente imagen:

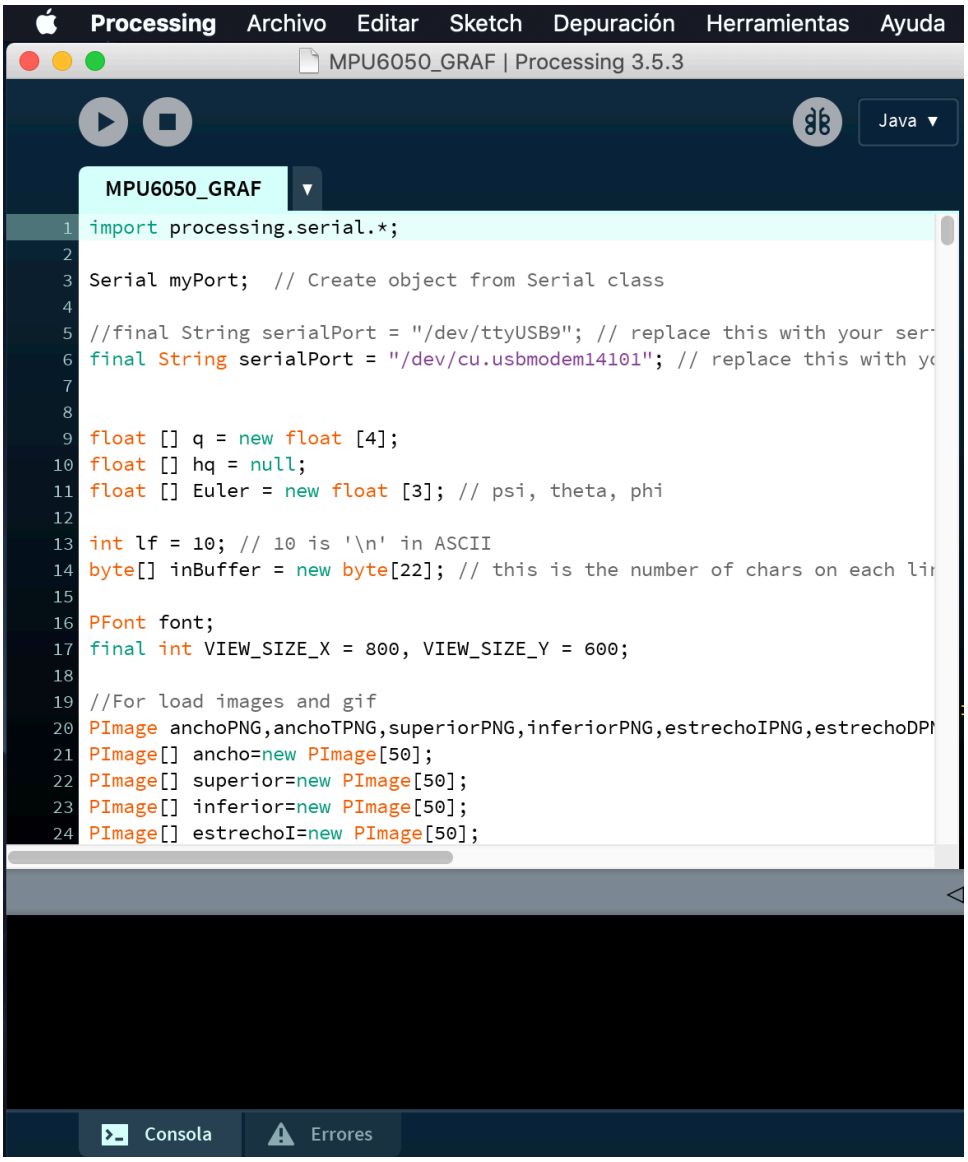



*Ilustración 23: IDE Processing*

Por último, una vez abierto dicho interfaz, para empezar a programar, es tan simple como empezar a escribir una línea de código acabada en “;” o varias si se va a hacer un programa complejo. Además, la mayoría de programas, al ser un lenguaje basado en Java, comienzan con un “import”, para importar las librerías y/o dependencias necesarias.

## 3.2 Tecnologías en el Sensor de movimiento ocular

### 3.2.1 Arduino

Esta tecnología es la misma que en las gafas VR, por lo que, este punto será el mismo que el 3.1.1 Arduino.

### 3.2.2 MATLAB

#### 3.2.2.1 MATLAB como lenguaje

MATLAB (.m) es un potente paquete de software para computación científica, orientado al cálculo numérico, a las operaciones matriciales y especialmente a las aplicaciones científicas y de ingeniería.

Puede ser utilizado como simple calculadora matricial, pero su interés principal radica en los cientos de funciones tanto de propósito general como especializadas que posee, así como en sus posibilidades para la visualización gráfica.

MATLAB posee además un lenguaje de programación propio, muy próximo a los habituales en cálculo numérico (Fortran, C, ...) que permite al usuario escribir sus propios scripts para resolver un problema concreto y también escribir nuevas funciones con, por ejemplo, sus propios algoritmos. MATLAB dispone, además, de numerosas Toolboxes, que le añaden funcionalidades especializadas.

Los scripts de MATLAB deben guardarse en un fichero con sufijo .m para ser reconocidos. Para ejecutar un script que esté en el directorio de trabajo, basta escribir su nombre (sin el sufijo) en la línea de comandos.

Una función (habitualmente denominadas M-funciones en MATLAB), es un programa con una "interface" de comunicación con el exterior mediante argumentos de entrada y de salida.

Por último, las funciones MATLAB responden al siguiente formato de escritura (la clausula end del final no es obligatoria, excepto en el caso de funciones anidadas, o sentencias de bucles):

```
function [argumentos de salida] = nombre(argumentos de entrada)
% comentarios
%
....
instrucciones (normalmente terminadas por ; para evitar eco en
pantalla)
....
end (opcional salvo en las funciones anidadas)
```

*Ilustración 24: Ejemplo funciones MATLAB*

#### 3.2.2.2 MATLAB como entorno

MATLAB es un programa para realizar cálculos numéricos con vectores y matrices. Como caso particular puede también trabajar con números escalares −tanto reales como complejos−, con cadenas de caracteres y con otras estructuras de información más complejas. Una de las capacidades más atractivas es la de realizar una amplia variedad de gráficos en dos y tres dimensiones.

Desde el punto de vista del control, MATLAB se puede considerar un entorno matemático de simulación que puede utilizarse para modelar y analizar sistemas. Permitir a el estudio de sistemas continuos, discretos, lineales y no lineales, mediante descripción interna y externa, en el dominio temporal y frecuencia. Matlab constituye un entorno abierto, para el cual numerosos paquetes específicos adicionales (*toolboxes*) han sido desarrollados. Estos paquetes específicos adicionales están constituidos por un conjunto de funciones que pueden ser llamadas desde el programa y mediante las cuales se pueden realizar multitud de operaciones.

Es un programa interactivo orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos. Además, es un programa command-driven, es decir, que se introducen las órdenes escribiéndolas una a una a continuación del símbolo ">>" (prompt) que aparece en una interfaz de usuario.

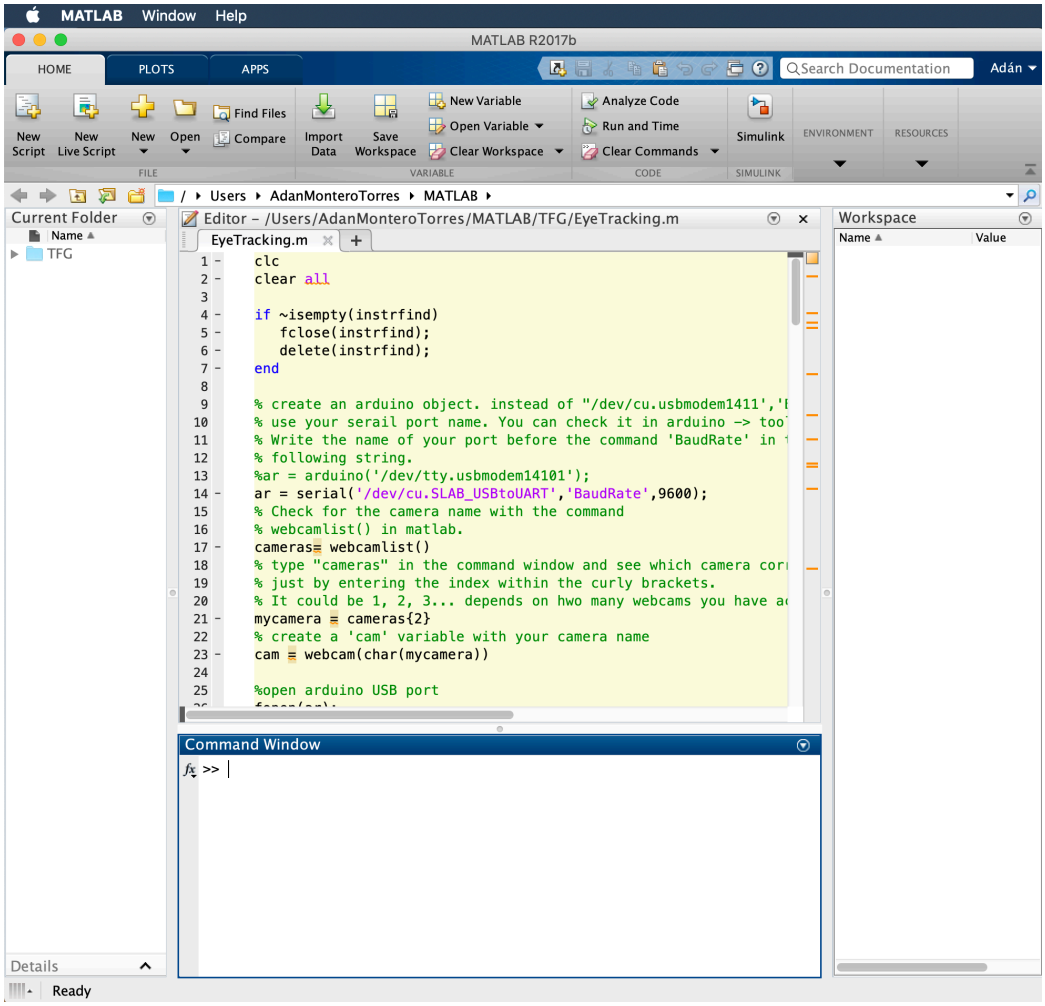



*Ilustración 25: Entorno MATLAB*

Por último, hablaremos del Workspace (espacio de trabajo) es el conjunto de variables que en un momento dado están definidas en la memoria del MATLAB.

Las variables creadas desde la línea de comandos de MATLAB pertenecen al base workspace (espacio de trabajo base). Los mismo sucede con las variables creadas por scripts que se ejecutan desde la línea de comandos. Estas variables permanecen en el base workspace cuando se termina la ejecución del script y se mantienen allí durante toda la sesión de trabajo o hasta que se borren.

# 4 SOFTWARE Y FUNCIONALIDAD

En este capítulo se detallará el software realizado con todas las tecnologías y herramientas explicadas anteriormente, además de la funcionalidad completa de la aplicación desarrollada.

## 4.1 Software y funcionalidades gafas VR

### 4.1.1 Software gafas VR

Cómo vimos en el capítulo anterior, el software de las gafas estará dividido en dos partes, el software para el MPU-6050 y Arduino y el software de Processing para simular el entorno en la Raspberry.

Tendremos dos códigos, para la parte de MPU/Arduino (.ino) y un código en la parte de la Raspberry (.pde), como vamos a ver a continuación.

#### 4.1.1.1 MPU_calibration.ino

Este código se encarga de calibrar las componentes espaciales, así como las aceleraciones del MPU-6050, en función de cómo lo coloquemos. Imprimirá por pantalla los valores de las aceleraciones X, Y, Z, además de las posiciones X, Y, Z, una vez que pulsemos un carácter.

```
-> Send any character to start sketch.
->
->
-> MPU6050 Calibration Sketch
->
-> Your MPU6050 should be placed in horizontal position, with package letters facing up.
-> Don't touch it until you see a finish message.
->
-> MPU6050 connection successful
->
-> Reading sensors for first time...
->
-> Calculating offsets...
-> ...
-> ...
-> ...
-> ...
-> ...
-> ...
->
-> FINISHED!
->
-> Sensor readings with offsets:   -6      -7      16379   -1      2       0
-> Your offsets:   -1800   -817    1001    11      8       16
->
-> Data is printed as: acelX acelY acelZ giroX giroY giroZ
```

*Ilustración 26: Calibración MPU-6050*

Una vez tengamos esos valores, podemos usarlos para configurar el dispositivo perfectamente en el código final de Arduino, como veremos más tarde.

Por último, veremos el diagrama de flujo de este código para lograr un mayor entendimiento, aunque el código completo lo podremos encontrar en el Anexo A.

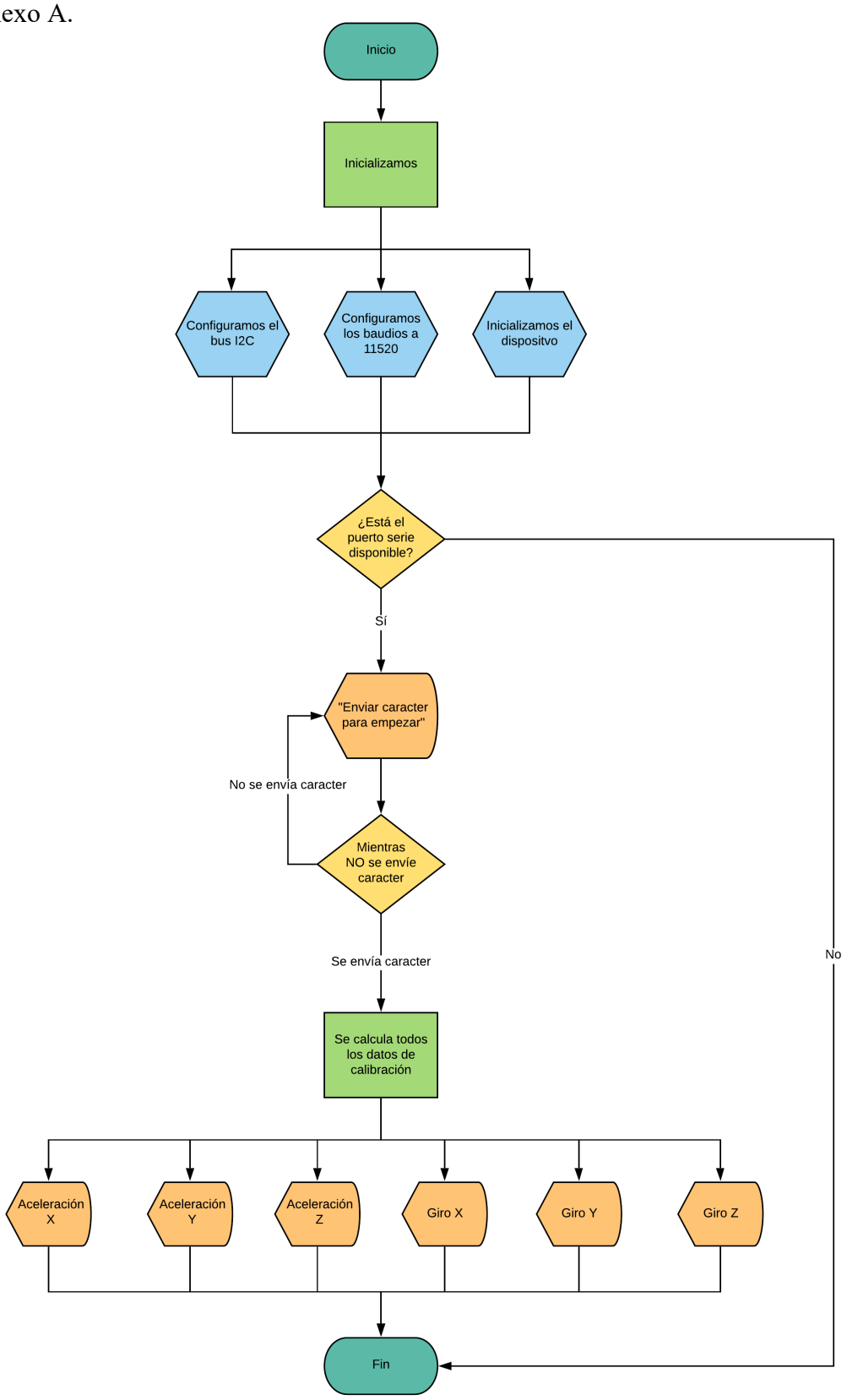



*Ilustración 27: Diagrama de flujo MPU_calibration*

#### 4.1.1.2 MPU_Arduino.ino

Este es uno de los códigos principales para hacer funcionar la aplicación, se encarga enviar por el puerto serie las componentes espacio y aceleración del dispositivo, en cuaterniones, unidad en la que el dispositivo proporciona los datos y la cual, vamos a proceder a explicar a continuación.

Los **cuaterniones** (también llamados cuaternios) son una extensión de los números reales similar a la de los números complejos. Mientras que los números complejos son una extensión de los reales por la adición de la unidad imaginaria *i*, tal que , los cuaterniones son una extensión generada de manera análoga añadiendo las unidades imaginarias *i*, *j* y *k* a los números reales tal que:

$$i^2 = j^2 = k^2 = ijk = -1$$

Los elementos *1*, *i*, *j* y *k* son los componentes de la base de los cuaterniones considerado como un $\mathbb{R}$-espacio vectorial de dimensión 4.

Después de esta pequeña introducción, si entramos un poco más en detalle en este código, en la siguiente imagen vemos como se envían los datos almacenados en un array por el puerto serie, que más tarde, será procesado en la Raspberry:

```
-> -0.83,0.53,-0.01,0.14,
-> -0.87,0.47,-0.01,0.15,
-> -0.90,0.41,-0.01,0.15,
-> -0.92,0.36,-0.01,0.15,
-> -0.94,0.31,-0.01,0.16,
-> -0.95,0.27,-0.01,0.16,
-> -0.96,0.23,-0.01,0.16,
```

*Ilustración 28: Cuaterniones enviados por el puerto serie*

De izquierda a derecha, lo que se envía es la aceleración Z, posición X, posición Y posición Z, en cuaterniones, que posteriormente serán transformados en ángulos.

Para entender todo mucho mejor, a continuación, se detallará el diagrama de flujo del código, aunque el código completo se detallará en el Anexo A.

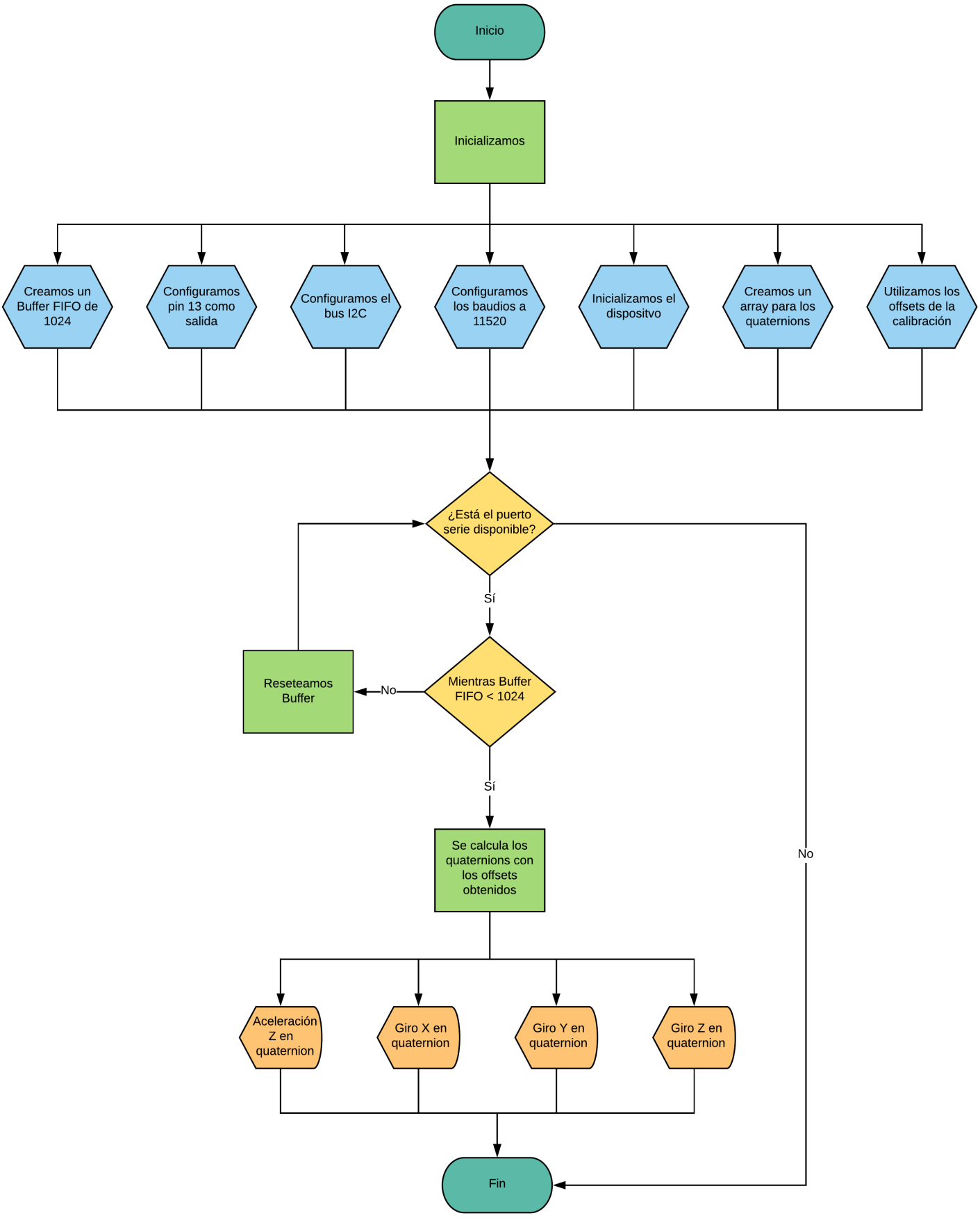



*Ilustración 29: Diagrama de flujo MPU_Arduino*

#### 4.1.1.3 MPU_Processing.pde

Antes de explicar y entrar en detalle en el código, vamos a hacer una breve mención al sistema operativo necesario para la correcta programación y funcionamiento de nuestro proyecto.

Hay muchas clases de sistemas operativos que funcionan en Raspberry, pero el que nos interesa es un sistema basado en Linux llamado **Raspbian**, ya que, es el más completo y versátil. Esta distribución se trata de una versión de Debian modificada y optimizada para sacar el máximo partido a este micro-ordenador. Raspbian se utiliza igual que Debian o que la mayoría de las distribuciones Linux populares, es muy sencilla, estable, rápida y podemos hacer con ella cualquier cosa instalando los programas que queramos.

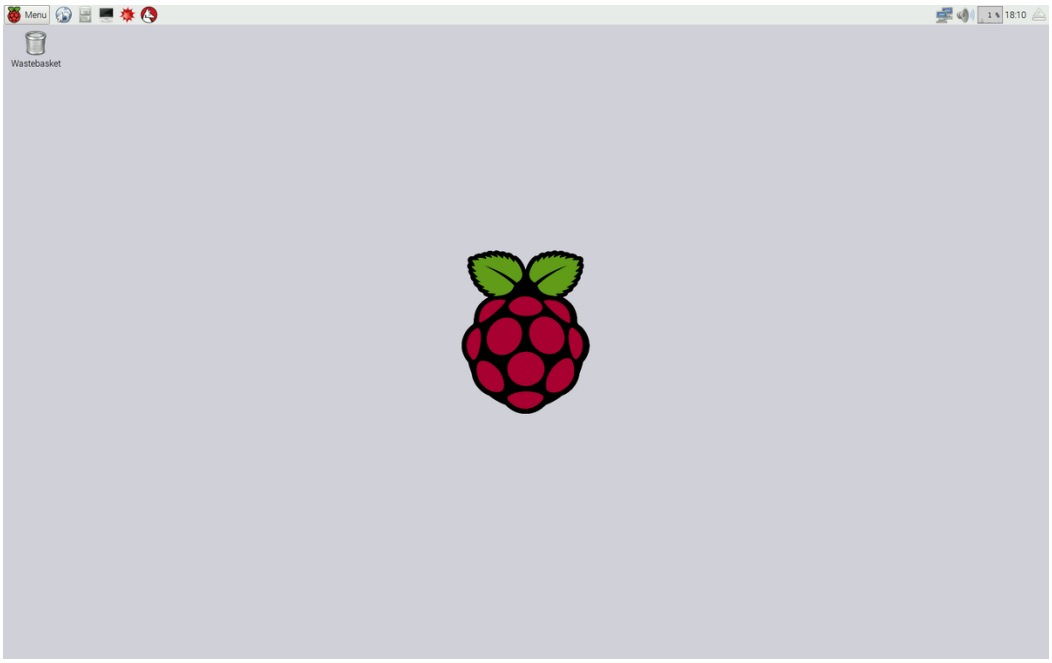



*Ilustración 30: Sistema Operativo Raspbian*

Por tanto, usaremos este sistema operativo, el cual, es capaz de ejecutar gran cantidad de softwares de programación, entre los cuales está el que nos interesa, Processing 3.

Después de esta pequeña introducción del sistema operativo de nuestra Raspberry Pi, procederemos a explicar en que consiste el código del programa encargado de hacer todo el entorno de realidad virtual con los datos que le llegan a través del puerto serie.

El código de este programa se podría dividir en dos partes, por un lado, tenemos la zona de código encargada de dibujar el entorno de realidad virtual, con un fondo y dos cubos con imágenes y por otro lado la parte que transforma los datos recibidos de cuaterniones a ángulos para que pueda dar el movimiento a los cubos.

Si nos centramos, primero, en esta última zona de código, pasaremos a explicar la transformación de cuaterniones a ángulo a través, de Euler.

Para empezar, las rotaciones espaciales en tres dimensiones se pueden parametrizar utilizando ángulos de Euler y cuaterniones de unidades, puede ser representada por un vector unitario y un ángulo de revolución alrededor de ese vector.

Cualquier rotación 3D se puede representar de esta manera, en otras palabras, dado un objeto sólido con orientación 1 y el mismo objeto con una orientación diferente 2. Entonces siempre podemos encontrar un eje y un ángulo que rotará de la orientación 1 a la orientación 2.

Entonces, tras esta pequeña introducción, podemos describir un cuaternio como:

$$\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T$$
$$|\mathbf{q}|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2 = q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$$

Esto, lo podemos asociar con la rotación de un eje mediante la siguiente expresión:

$$\mathbf{q}_0 = \mathbf{q}_w = \cos(\alpha/2)$$
$$\mathbf{q}_1 = \mathbf{q}_x = \sin(\alpha/2)\cos(\beta_x)$$
$$\mathbf{q}_2 = \mathbf{q}_y = \sin(\alpha/2)\cos(\beta_y)$$
$$\mathbf{q}_3 = \mathbf{q}_z = \sin(\alpha/2)\cos(\beta_z)$$

Donde α es un ángulo de rotación simple (el valor en radianes del ángulo de rotación) y cos (βx), cos (βy) y cos (βz) son los "cosenos de dirección" que ubican el eje de rotación (teorema de rotación de Euler).

De manera similar para los ángulos de Euler, usamos los ángulos de Tait Bryan:

- Φ: phi, rotación sobre el nuevo eje X.
- θ: theta, rotación sobre el nuevo eje Y.
- Ψ: psi, rotación sobre el nuevo eje Z.

Donde el eje X apunta hacia adelante, el eje Y hacia la derecha y el eje Z hacia abajo con ángulos definidos para la rotación hacia la derecha / izquierda.

Entonces, finalmente, los ángulos de Euler pueden se pueden obtener con la siguiente relación:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan\frac{2(q_0 q_1 + q_2 q_3)}{1-2(q_1^2+q_2^2)} \\ \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \arctan\frac{2(q_0 q_3 + q_1 q_2)}{1-2(q_2^2+q_3^2)} \end{bmatrix}$$

Sin embargo, tenga en cuenta que las funciones arctan y arcsin implementadas en lenguajes de computadora solo producen resultados entre $-\pi / 2$ y $\pi / 2$, y para tres rotaciones entre $-\pi / 2$ y $\pi / 2$ no se obtienen todas las orientaciones posibles. Para generar todas las orientaciones, uno necesita reemplazar las funciones arctan en el código de la computadora por atan2:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \operatorname{asin}(2(q_0 q_2 - q_3 q_1)) \\ \operatorname{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

Para terminar, hay que tener en cuenta las singularidades en la parametrización del ángulo de Euler cuando el tono se aproxima a $\pm 90°$ (polo norte / sur). Estos casos deben ser manejados especialmente.

Una vez explicado toda la teoría, podemos pasar a profundizar en el código, el cual, al menos esta parte, es básicamente aplicar dichas ecuaciones después de haber leído los datos recibidos por el puerto serie.

En cuanto a la parte de dibujar, primero, se inicializan todas la variables y se comprueba si el puerto está disponible en la función principal setup (), todo lo demás, se hace en la otra función principal, draw (), donde se establece un fondo, ya sea un color o una imagen y, además, en este caso para dibujar los cubos, se ha hecho uso de una función llamada drawcube (), donde se dibujan ambos cubos además de establecer las imágenes.

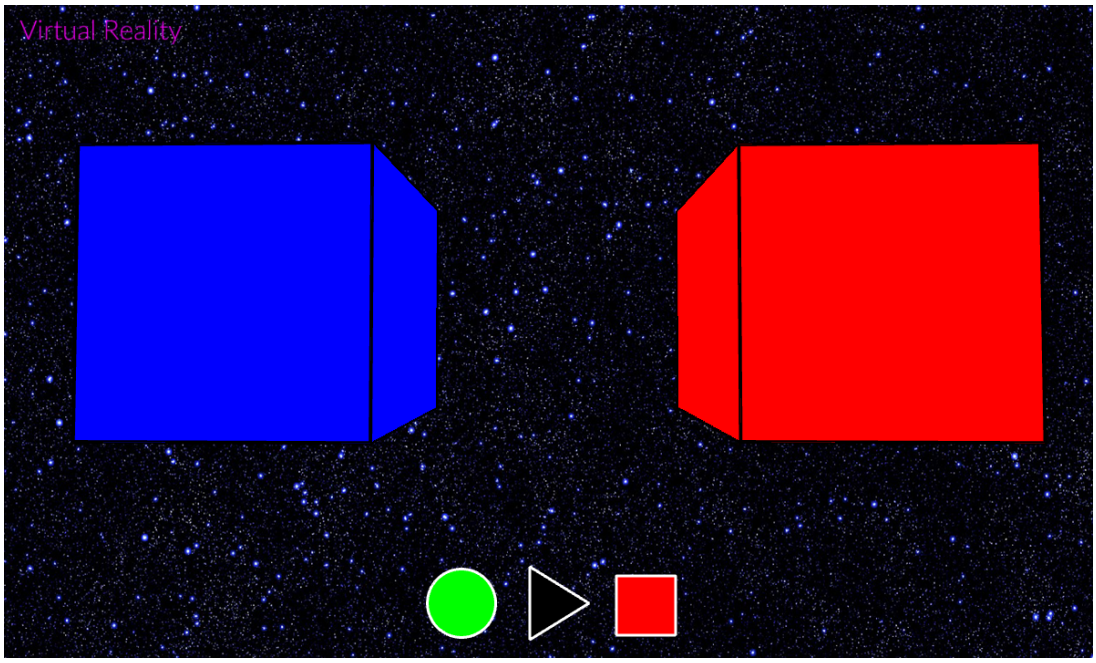



*Ilustración 31: Resultado final código Processing*

Como contenido adicional, se ha añadido unos botones de iniciar (círculo), siguiente, para cambiar las fotos en los cubos (triángulo) y parar (cuadrado) en la parte baja central del entorno.

Para que todo quede aún más claro, a continuación, se adjunta el diagrama de flujo de todo el programa. El código completo estará en el Anexo A.

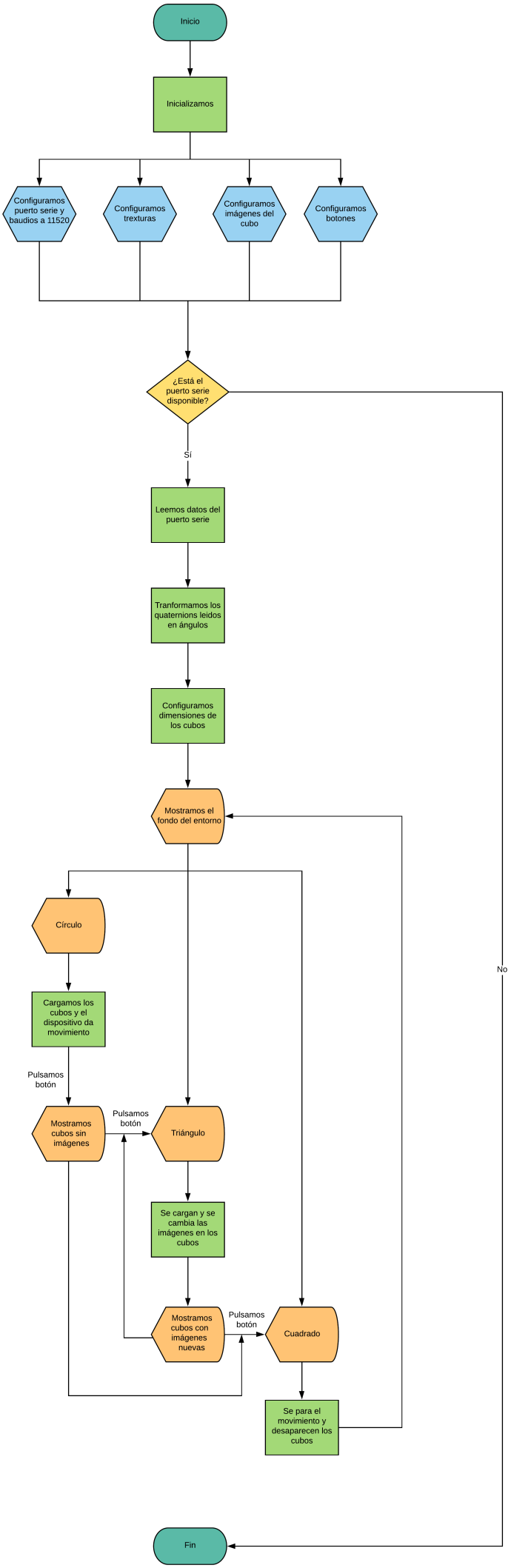



*Ilustración 32: Diagrama de flujo MPU_Processing*

### 4.1.2 Funcionalidades gafas VR

Una vez explicado todo el software, es hora de ver todas las funcionalidades que hemos creado en el proyecto.

Nada más iniciar la aplicación de las gafas nos encontraremos con esta interfaz que durará unos segundos:


*Ilustración 33: Splash Screen aplicación*

Tras esto, aparecerá la interfaz vacía con los tres botones antes mencionados:

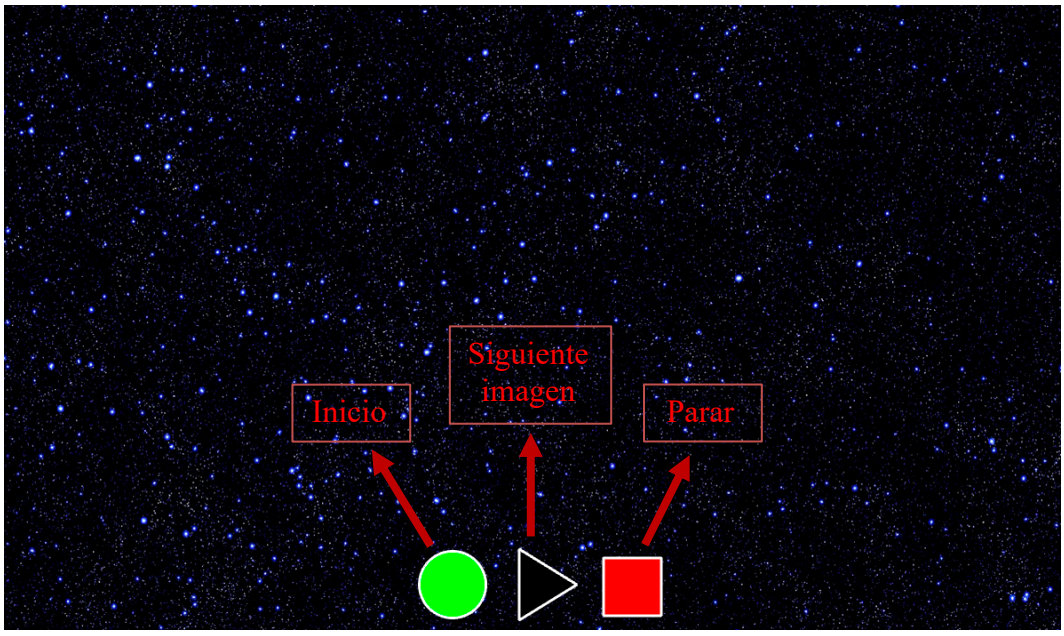



*Ilustración 34: Interfaz vacía*

Si pulsamos el círculo verde, se inicia la realidad virtual y aparecen los dos cubos que lo simulan:

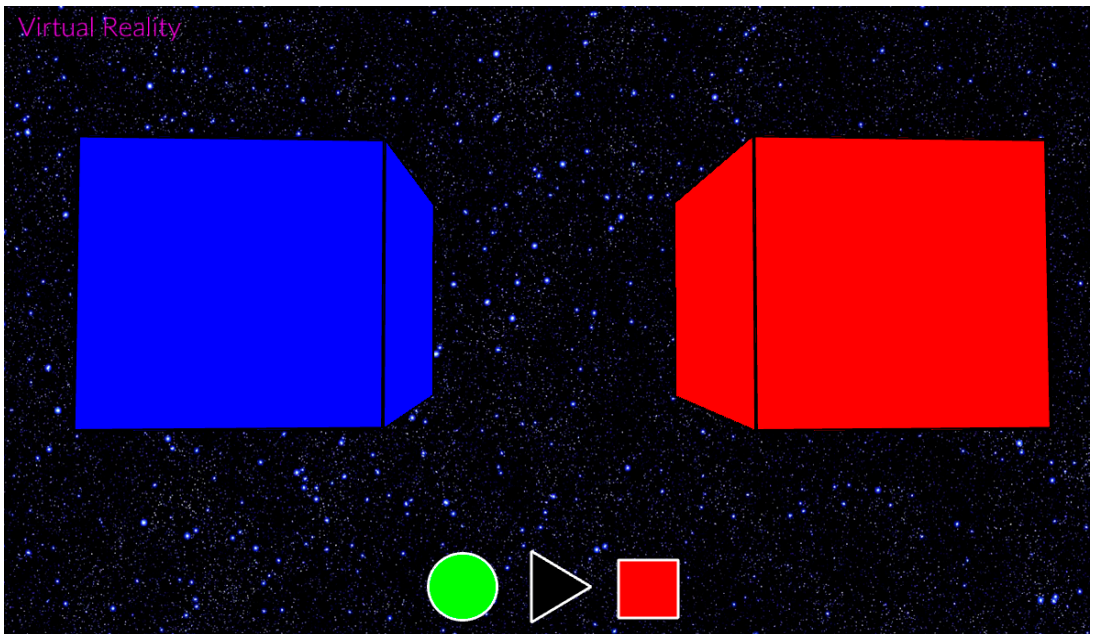



*Ilustración 35: Primera interfaz*

Si pulsamos el triángulo, siguiente, se cambiarán las imágenes de los cubos:

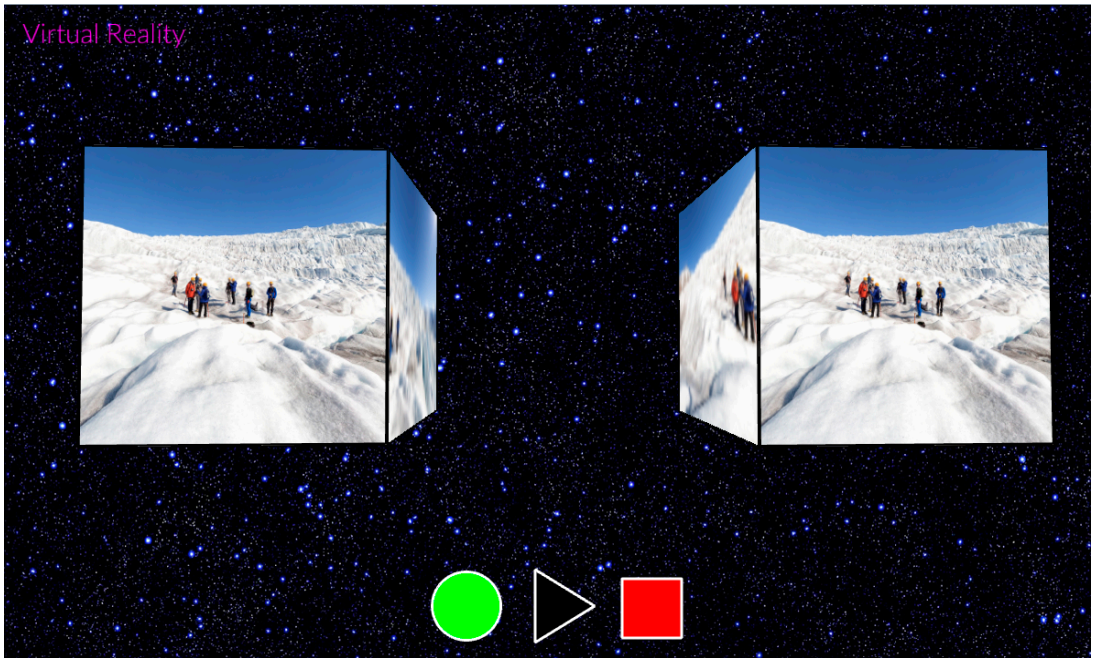



*Ilustración 36: Interfaz con imágenes*

Cuando tengamos la imagen deseada, procederemos hacer el movimiento a través del MPU, como podemos ver en la siguiente imagen:

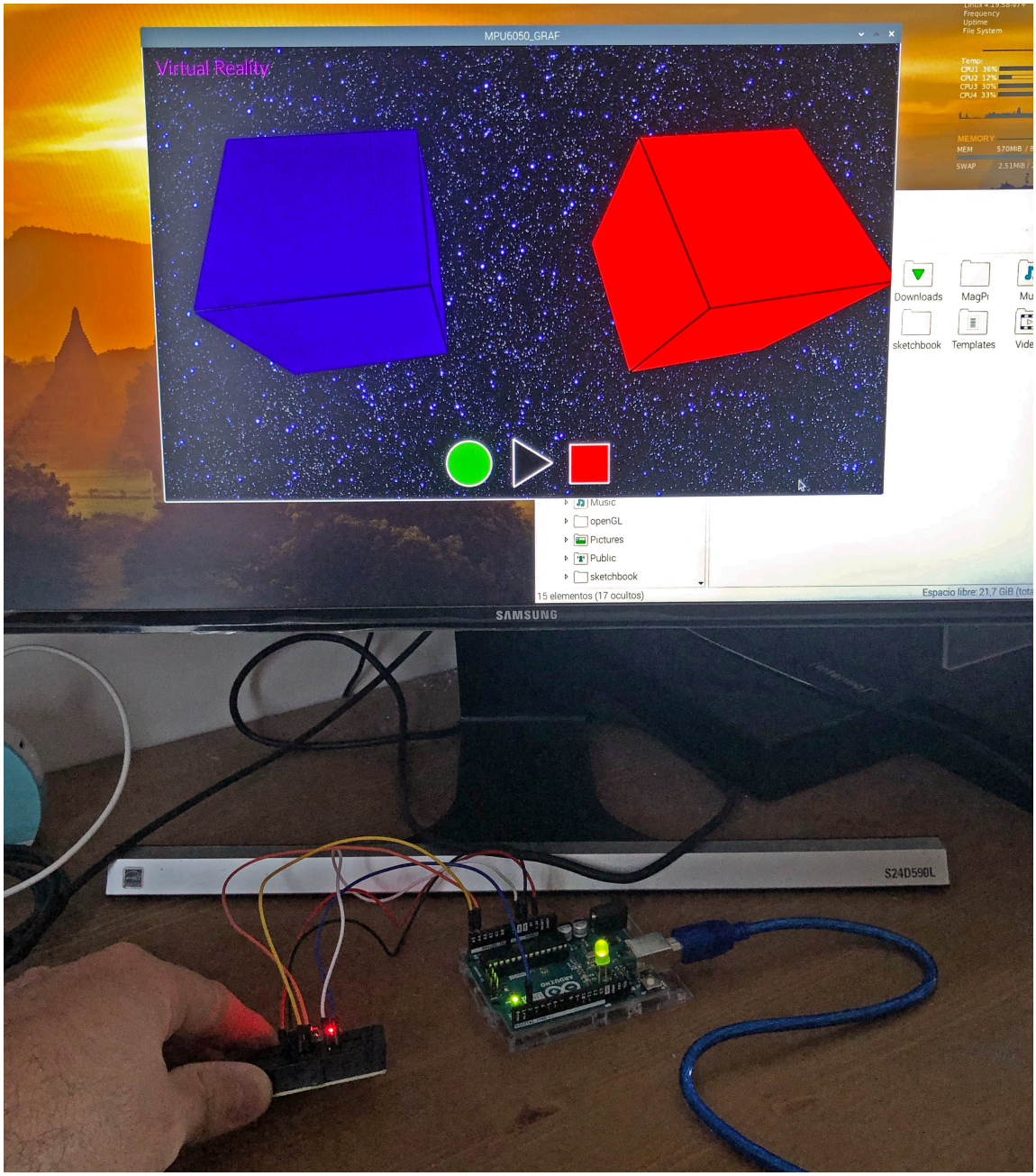



*Ilustración 37: Movimiento entorno VR*

En el caso de que cuando se inicie la aplicación y se quieran realizar algunos movimientos, la posición del MPU no coincide con la posición de los cubos como en la imagen:

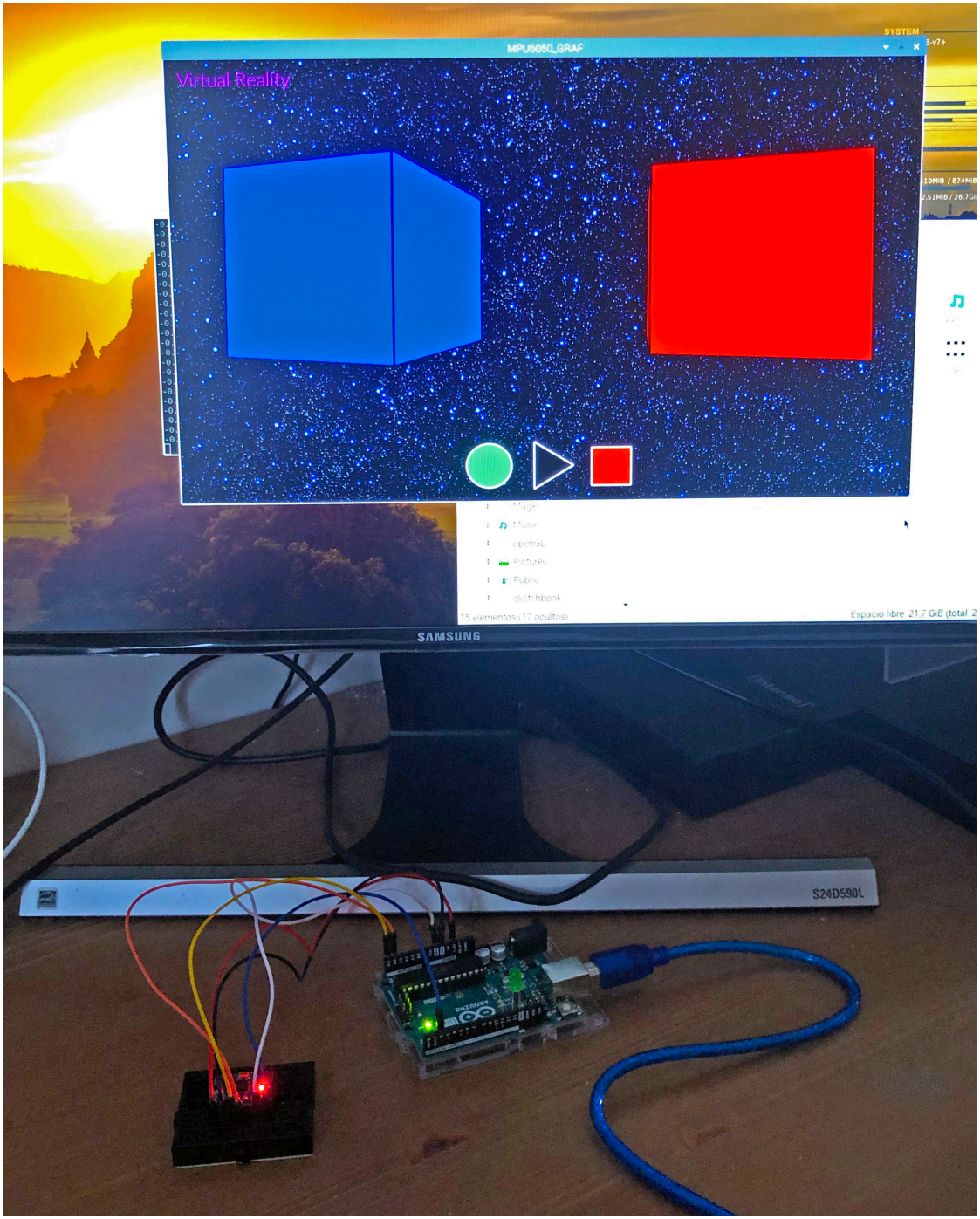



*Ilustración 38: Movimiento sin coincidir*

Por ello, al tener pequeños errores de offsets se ha programado en el código una pequeña corrección, que será pulsando la tecla “h” del teclado el MPU totalmente recto, como podemos ver en la imagen:

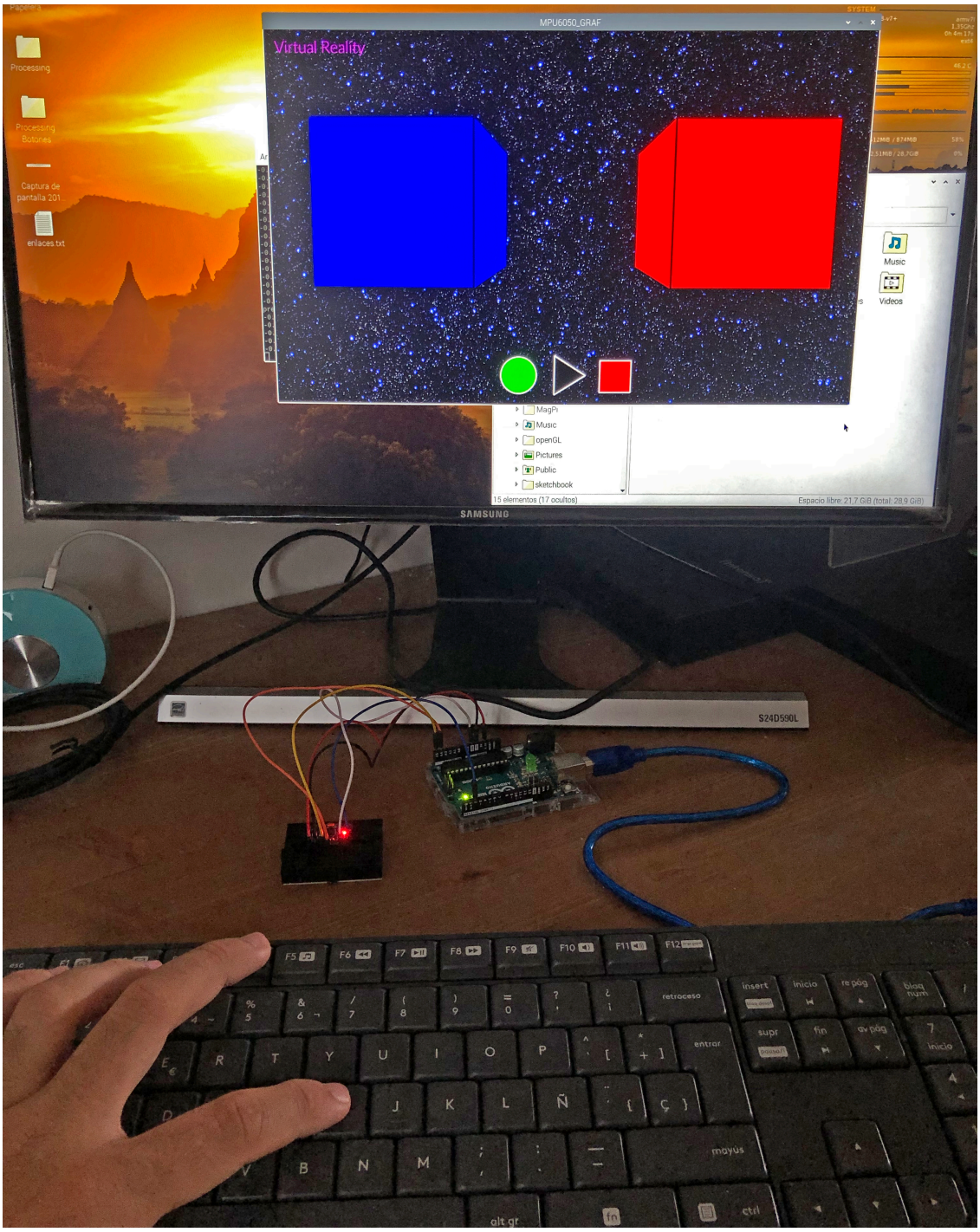



*Ilustración 39: Ajuste posición entorno*

Se puede ver como la posición coincide perfectamente con el MPU y se ha corregido el error anterior.

Por último, se podría seguir pulsando al botón siguiente para ir cambiando imágenes, pero si lo que queremos es parar, si pulsamos sobre el botón cuadrado, el entorno dejará de aparecer y será como cuando comienza:

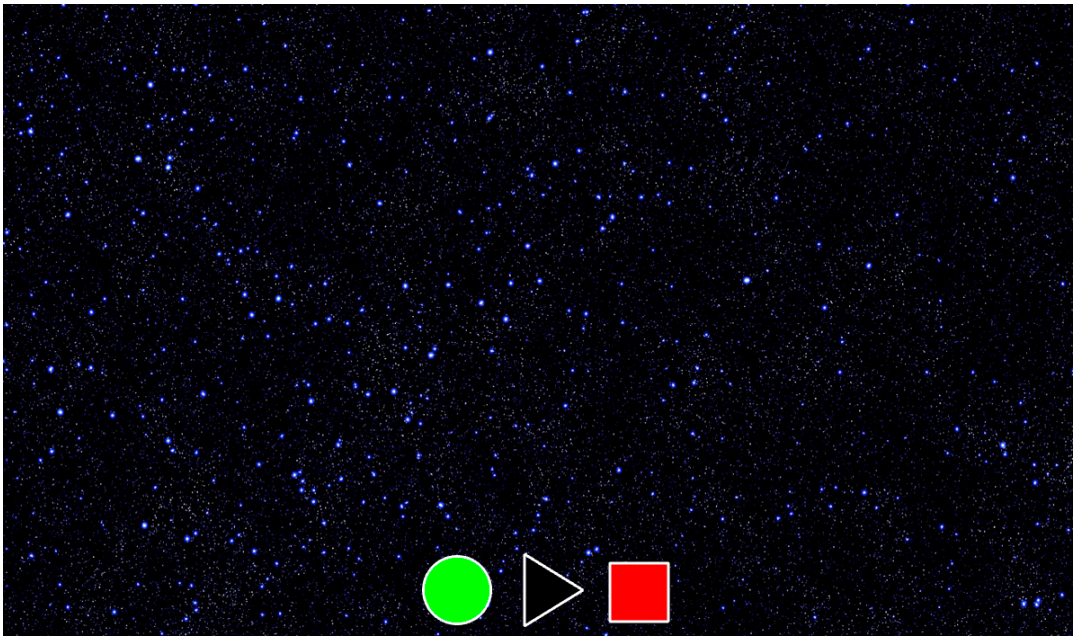


*Ilustración 40: Interfaz vacía tras pulsar el botón de parar*

Y con esto, se terminaría la simulación del entorno de realidad virtual, siendo posible acceder a él, iniciando, cambiando imágenes y parando las veces que hicieran falta.

## 4.2 Software y funcionalidades del sensor ocular

### 4.2.1 Software del sensor ocular

Tal y como vimos cuando hablamos del sensor, este tiene dos tipos de códigos, el de Arduino (.ino) y el de MATLAB (.m), y solo tendrá un código por cada una de las tecnologías, que pasaremos a detallar cada uno de ellos a continuación.

#### 4.2.1.1 Ardunio_MATLAB_Serial.ino

Este código es muy simple, lo único de lo que se encarga, es de hacer la comunicación serie con MATLAB y configurar el pin donde se conectará los leds infrarrojos.

Puesto que no tiene mucho más detalle, se detallará el diagrama de flujo para que todo quede más claro, además de que el código completo estará en el Anexo A.

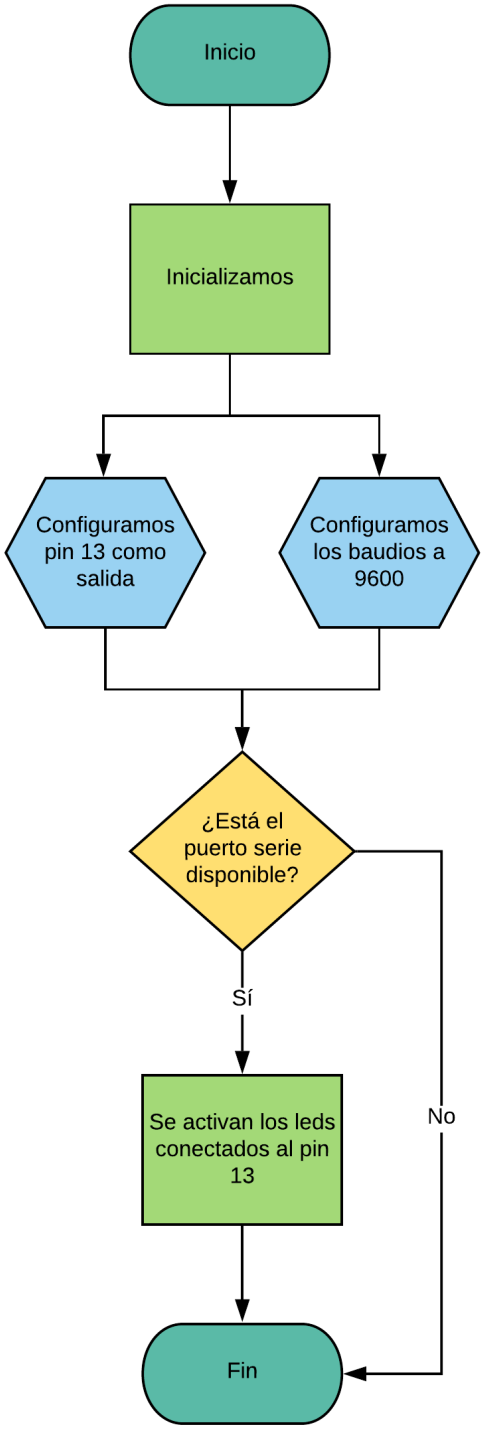



*Ilustración 41: Diagrama de flujo Ardunio_MATLAB_Serial*

#### 4.2.1.2 EyeTracking.m

Este código [10, 13] sí es algo más complejo que el anterior, pero aún así, no tiene mucha complicación, como veremos. Entre lo que más destaca, es que una vez que se comprueba que el puerto serie está disponible, este listará las cámaras que están conectadas, a través del comando, "*webcamlist*" al ordenador donde se está ejecutando, puesto que puede tener una cámara interna y esta estará visible como cámara 1, por lo tanto, como nosotros conectaremos una cámara externa (aunque modificada), aparecerá como cámara 2, y esta será la que se usará.

```
>> webcamlist

ans =

  2×1 cell array

    {'FaceTime HD Camera'}
    {'Trust Webcam'      }
```

*Ilustración 42: Listado de cámaras conectadas*

Una vez elegida y conectada la cámara, lo siguiente que se hará es encenderla y que se vea a través del comando "preveiew(cam)", pero al tener conectada los leds infrarrojos, se verá todo en blanco y negro y no hará falta nada de luz, de hecho, cuantas menos luces, menos sombras y por lo tanto mejor se verá el ojo.


*Ilustración 43: Ojo en directo*

Por último, una vez que se inicia la visualización del ojo, se abre otra pantalla, la cual, se encarga de ir fotografiando cada movimiento del ojo y captando la pupila de este, subrayándola en rojo, como veremos a continuación.

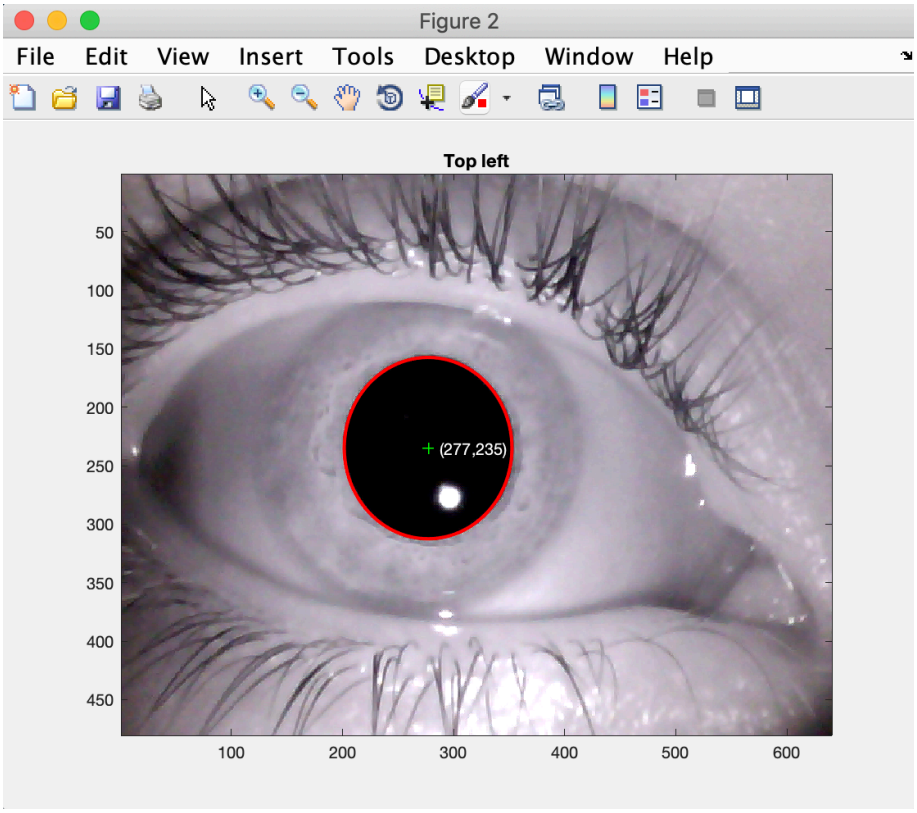



*Ilustración 44: Fotografía de la pupila captada*

Se puede observar, que en función de donde la pupila mire, el título de la foto variará. Se podrá distinguir 4 posiciones: Arriba izquierda, Arriba derecha, Abajo izquierda y Abajo derecha.

El número de fotografías variará en función de lo que se programe, puede ser desde 1 hasta las que se necesite, ya que es un bucle for. Estas fotos se almacenarán en la memoria interna del ordenador en el que se ejecuta.

Para una mejor comprensión del código a continuación, se detallará el diagrama de flujo, aunque, como los demás, el código completo estará en el Anexo A.

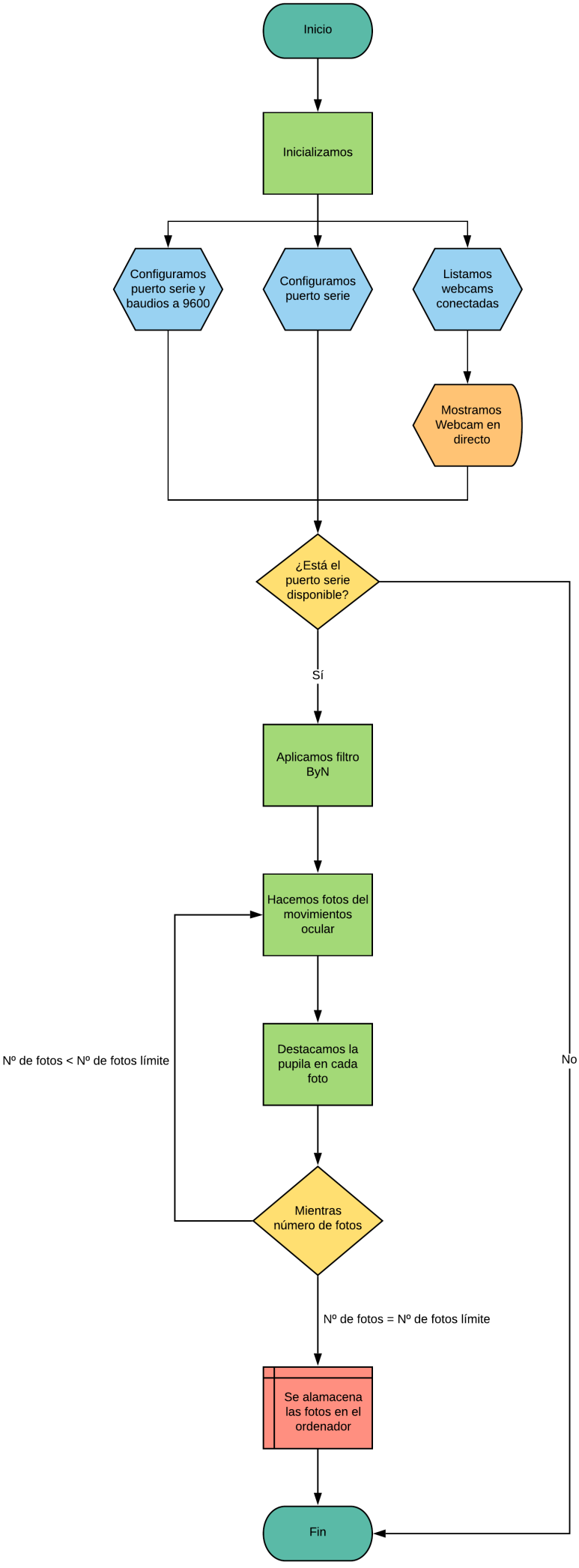



*Ilustración 45: Diagrama de flujo EyeTracking*

### 4.2.2 Funcionalidades del sensor ocular

Aunque ya se haya visto gran parte de la funcionalidad mientras se explicaba el software, en este apartado, se profundizará un poco más.

Para empezar, el soporte de la cámara, por comodidad, funcionalidad y parecido a la realidad, estará en un soporte lo más parecido posible a unas gafas de realidad virtual.

*Ilustración 46: Parte frontal soporte sensor ocular*

*Ilustración 47: Parte interior soporte sensor ocular*

Como se puede ver, se ha intentado hacer lo más firme y ergonómica posible puesto que el ojo debe estar fija en una posición sin que sufra movimientos.

Una vez puesto, se conectará tanto el cable de la cámara (negro) como los cables que van de los leds infrarrojos a la protoboard y de este al Arduino Pro Mini, como podemos ver en la siguiente imagen:

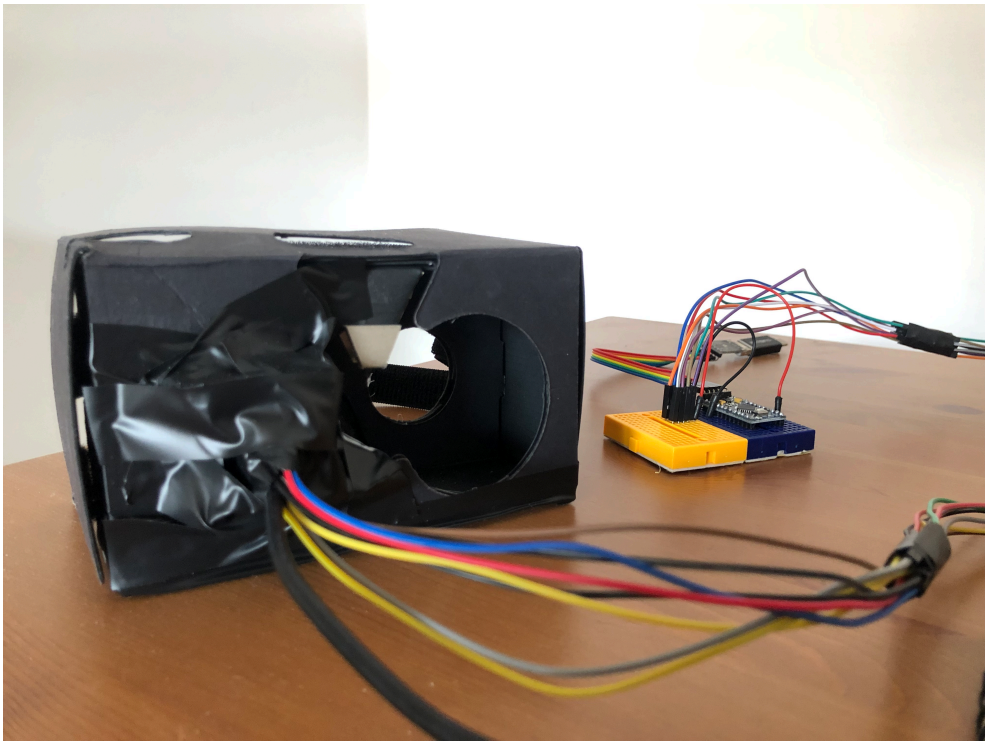


*Ilustración 48: Cables del sensor ocular*

Una vez conectado, se pasará a ejecutar el código de MATLAB y este abrirá dos ventanas, como ya hemos visto anteriormente, una en la que se ve el ojo en directo, y otra en la que se ve las fotografías que se sacan al ojo, con la captación de la pupila, en rojo.

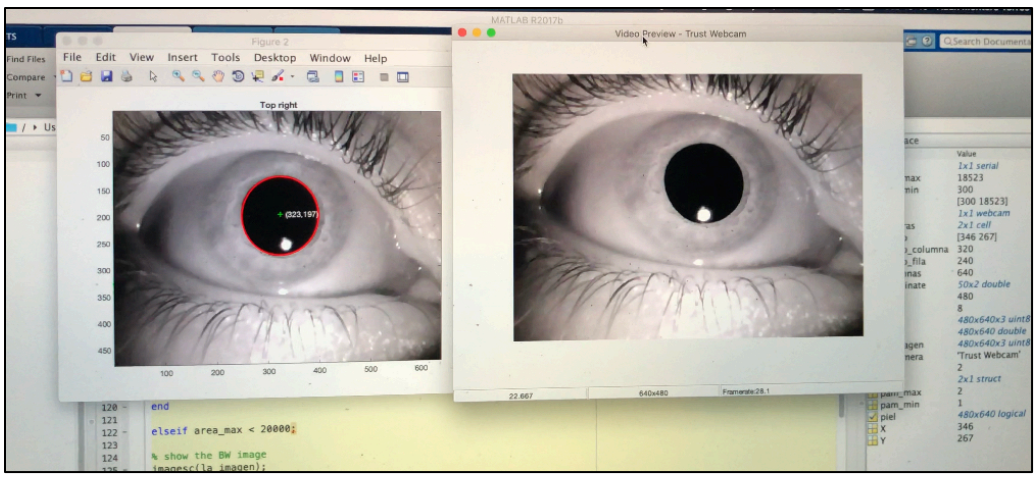



*Ilustración 49: Ejecución código en MATLAB*

Por último, mientras el código se está ejecutando, esas imágenes se irán guardando a la ruta que yo le haya indicado, que, en este caso, será en la misma carpeta donde está el código EyeTracking.m.

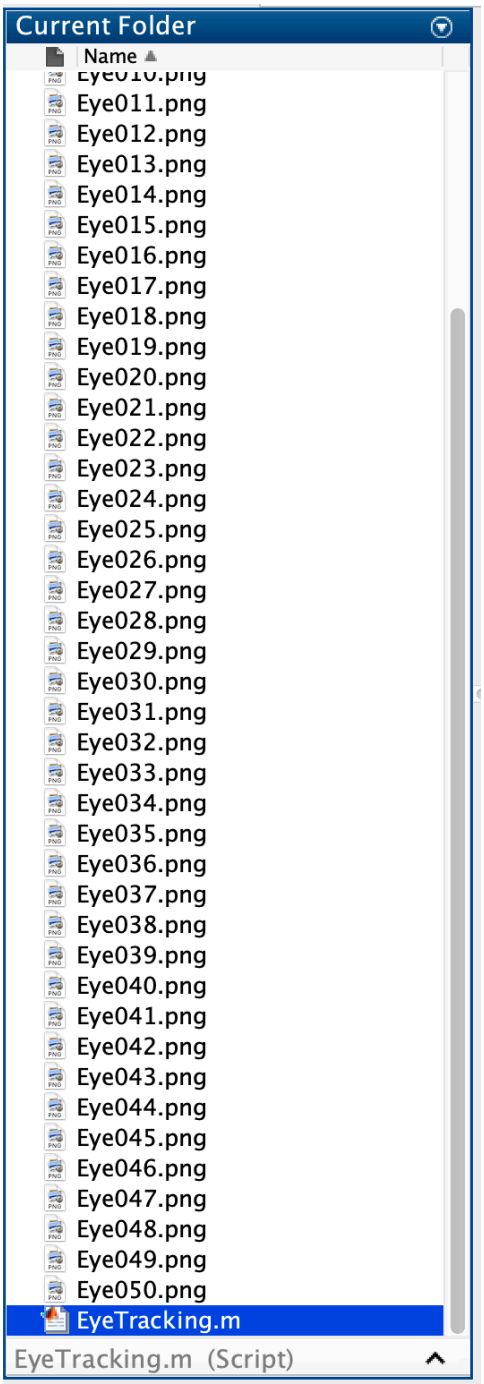



*Ilustración 50: Directorio donde se guardan las imágenes*

En este caso, se han guardado 50 fotografías, puesto que el bucle for estaba configurado para que hiciera ese número de fotos.

# 5 PROBLEMAS ENCONTRADOS

Este quinto capítulo se centrará en describir algunos problemas obtenidos durante el desarrollo del proyecto. Dichos problemas son tanto a nivel software como a nivel hardware en distintas partes del proyecto, como veremos a continuación.

## 5.1 Rendimiento Raspberry

Uno de los mayores problemas de esta placa es el sobrecalentamiento, puesto que al tener que mover sistemas gráficos tan pensados (además del sistema operativo elegido) necesita un buen sistema de refrigeración. Al no tenerlo, el rendimiento de esta placa se ve afectado haciendo que todo el sistema vaya más lento y en varias ocasiones se quedase colgado debido a las altas temperaturas que llegan a alcanzar algunas partes de la placa.

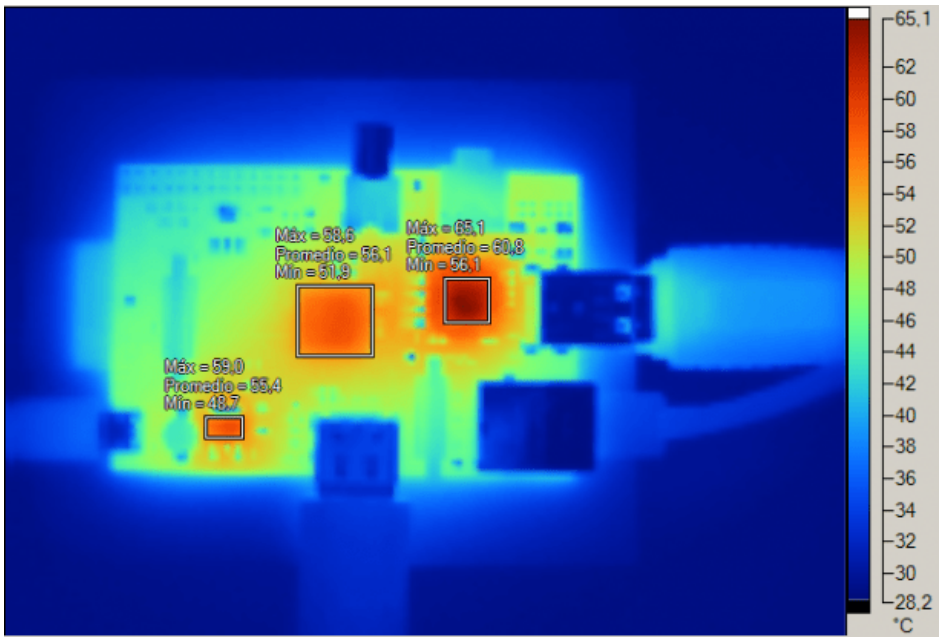



*Ilustración 51: Temperatura Raspberry sin sistema de refrigeración*

La solución a todo esto, es ponerle en esos puntos de la placa unos disipadores especiales de dichos tamaños y con distintos tipos de materiales, como los siguientes:

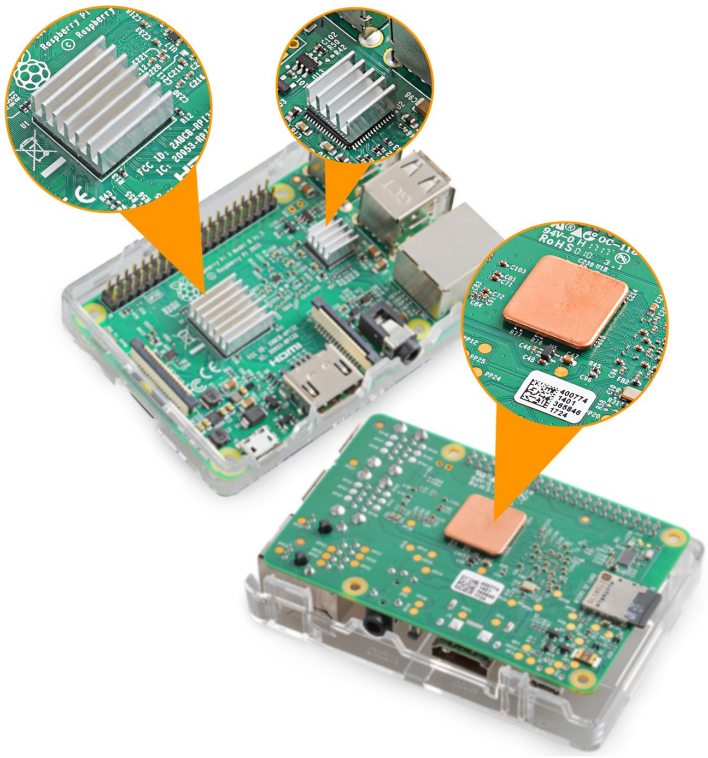


*Ilustración 52: Disipadores especiales Raspberry*

Además, existe la opción de ponerle una caja con un pequeño ventilador, para así tener mejor refrigeración aún, haciendo que la temperatura baje aún más:

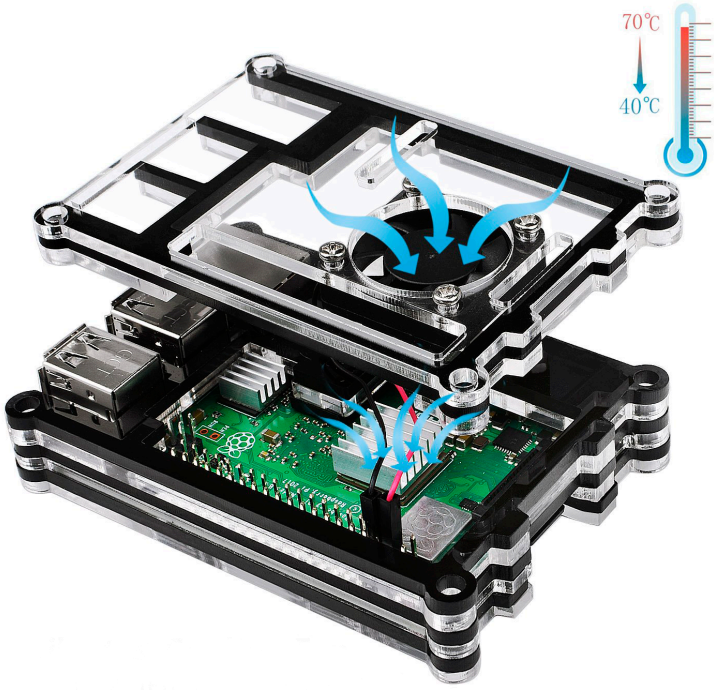



*Ilustración 53: Sistema de refrigeración completo*

## 5.2 Sensor ocular en la Raspberry

Una de las ideas principales era que el sensor ocular o Eye Tracking estuviera en la misma pantalla que la realidad virtual, para que se viera a la misma vez como se mueven los cubos y el ojo.


*Ilustración 54: Entorno VR con sensor ocular*

Aunque, después de hacer pruebas, se vio que el rendimiento del entorno se veía afectado por sobrecargarlo con dos comunicaciones serie distintas y no se apreciaba nada en el objeto destinando a ser sensor ocular, pues no se veía a penas el movimiento que realmente se buscaba, así que se descartó idea y se optó por buscar otra solución.

## 5.3 Luminosidad en el sensor ocular

Tras haber descartado la primera idea para el sensor ocular, se decidió por usar el sensor actual explicado en el proyecto. Una vez todo construido e instalado, cuando se hacían pruebas, se veía que el directo de la cámara no se veía en blanco y negro (cómo se debía de ver con los sensores infrarrojos), sino que se veía todo oscuro puesto que los sensores infrarrojos no estaban funcionando correctamente.

Además de que las fotos no se hacían bien, no se captaba bien la pupila (incluso teniendo una luz artificial) y se quedaba colgado.

Una vez comprobado la conexión de los leds infrarrojos, se vio como el problema era que no estaban recibiendo corriente y los leds no se iniciaban cuando el puerto serie hacía la conexión con MATLAB. Tras arreglarlo, todo funcionó a la perfección como se ha visto en el proyecto.

## 5.4 OpenGL

En un principio, el lenguaje de programación a usar para realizar el entorno de realidad virtual iba a estar basando en OpenGL, es decir, programar, por ejemplo, en lenguaje C, usando las librerías y dependencias de OpenGL.

El problema principal de todo esto fue que la programación para realizar un entorno, como el que finalmente se ha hecho, en este lenguaje con OpenGL era muchísimo más engorroso y complicado que hacerlo con la tecnología que finalmente se hizo, Processing, sobre todo a la hora de leer y procesar los datos recibidos por el puerto serie, pues había muchísimos problemas de compatibilidad y a la hora de leer los datos.

Además, aunque el lenguaje de Processing y OpenGL sea muy parecido, el entorno y la programación de Processing es mucho más limpia y menos engorrosa que la de OpenGL, en la cual, era mucho más tedioso el programar los diseños en 3D.

# 6 LÍNEAS FUTURAS

En este penúltimo capítulo se describirán algunas líneas de mejora que podrían seguirse en un futuro a la hora de mejorar el proyecto desarrollado y añadirle más funcionalidades con el fin de enriquecer la aplicación creada y aumentar sus posibilidades de éxito en el mercado actual.

En primer lugar, se hablará de algunas mejoras respecto a la parte hardware del sistema. A continuación, se destacan las siguientes:

- Para empezar, una de las mejoras más directas es la de conectar la parte de la realidad virtual, la de la Raspberry, a una pantalla portátil de 5 pulgadas para que todo pueda ser un sistema totalmente portátil y transportable, como unas gafas VR totalmente funcionales.

- Además de esto, añadirle unas lentes a dichas gafas para que se pueda centrar la vista bien en la realidad virtual programada.

- Otra posible mejora posible, sería añadir a dichas gafas unos auriculares o cascos envolventes capaces de soportar audio 3D/Bilateral en sustitución de los altavoces, además de que pueda reproducir lo que el psicólogo/terapeuta quiera hablarle

- Rediseñar por completo las gafas y ponerle el eslogan diseñado para esta.

- Otra de las opciones posibles sería el añadir un sistema más potente que la Raspberry Pi 3B+, el cual, fuera un sistema con mayor capacidad para poder soportar además de imágenes, mayores resoluciones y videos.

Por otro lado, otra posible mejora sería en la parte del software del sensor ocular, en donde este, se podría perfeccionar a la hora de que cuando se parpadee no se hiciera ninguna foto o que incluso pudiese medir la velocidad con la que la pupila se mueve en función de lo que esté reaccionando.

Todas estas posibles mejoras harían al actual proyecto algo revolucionario en el mercado actual, además de que sería una estupenda forma de hacer terapias para poder resolver traumas y conflictos.

# 7 PLANIFICACIÓN Y PRESUPUESTO

## 7.1 Planificación

El presente proyecto ha tenido varias fases en la planificación desde que comenzó en febrero de 2019 hasta su finalización en septiembre de este mismo año.

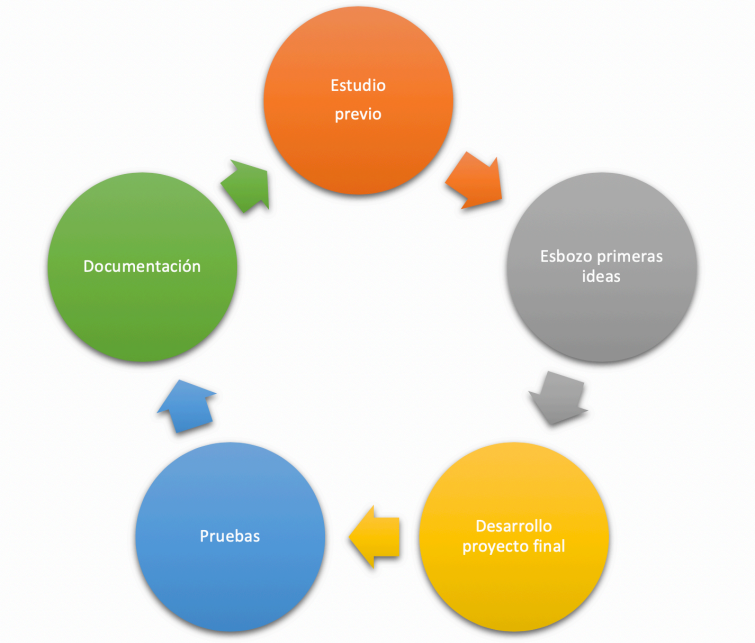



*Ilustración 55: Fases de planificación del proyecto*

Además, cada una de estas partes ha tenido una estimación diferente, puesto que unas han ocupado más horas que otras, como vemos a continuación:

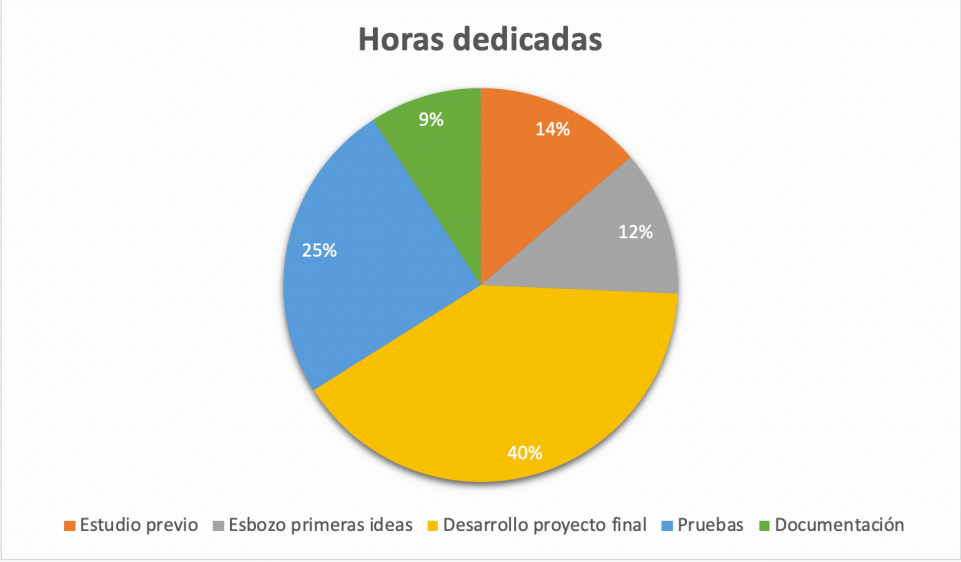



*Ilustración 56: Distribución de horas en el proyecto*

Como podemos observar, de los 7 meses que dura el proyecto, la mayor parte, casi 3 meses, serán para el desarrollo del proyecto final. Seguido de este, las pruebas para su correcto funcionamiento, que se estima que duren 1 mes y medio. Y, por último, repartiéndose el resto de tiempo de los 7 meses, el estudio previo, esbozo de primeras ideas y documentación.

## 7.2 Presupuesto

En este apartado se desglosará cada uno de los componentes utilizados con su coste, además de eso, se detallará el cómputo de horas totales con su coste y por último el coste total del proyecto.

| COMPONENTES | PRECIO | PROVEEDOR |
|---|---|---|
| | | |
| Kit Raspberry Pi 3B+ | 79,99 € | Amazon |
| Arduino UNO | 19,90€ | Amazon |
| Arduino Pro Mini | 5,72€ | eBay |
| Leds infrarrojos | 7,99€ | Amazon |
| Componentes Arduino | 20,99€ | Amazon |
| Disipadores Raspberry | 9,99€ | Amazon |
| Herramientas de soldadura | 23,99€ | Amazon |
| Webcam | 13,99€ | El Corte Ingles |
| Mano de obra Ingeniero (20€/hora) | 12.000€ | |
| TOTAL | **12.182,56€** | |

*Ilustración 57: Presupuestos de todo el proyecto*

Si desglosamos la mano de obra, se dice que un ingeniero cobra 20€/hora, entonces, de los 7 meses que dura el proyecto, que serían 196 días, de los cuales, entre fines de semana y festivos, se estima que se trabajará unos 100 días, 6 horas diarias, por tanto:

$$20€/h * 6h/días * 100\ días = 12.000€$$

# 8 CONCLUSIONES

Volviendo la vista hacia el principio del presente documento, resulta primordial para concluirlo, recordar en qué ha consistido el proyecto, a grandes rasgos.

El presente proyecto desarrollado es una aplicación software y hardware que hace uso de varias tecnologías para su correcto funcionamiento, para que sea una aplicación eficaz, rápida y por supuesto, para que sea una herramienta capaz de ayudar a otras personas a enfrentarse a sus traumas, conflictos o miedos.

Teniendo presente lo anterior, se pueden sacar algunas conclusiones finales sobre la trayectoria seguida, las dificultades encontradas a lo largo del desarrollo del y, sobre todo, el grado de satisfacción en cuanto al resultado final de este proyecto.

Se ha desarrollado una aplicación fácil de usar y comprender, destinada a cualquier clínica psicológica y/o terapeuta, que desee de realizar un tipo de terapia novedosa de la técnica de EMDR, la cual, ya por si sola tenía mucha aceptación y eficacia, junto a esta aplicación será capaz de aumentar su grado de eficiencia y con cierta proyección hacia un futuro comercial de la misma.

El tener unas gafas de realidad virtual hacen de esta aplicación una plataforma bastante llamativa, atractiva y realmente útil para cualquier persona con la necesidad y/o el deseo de tratarse de ese trauma reprimido o de esa sensación de malestar que no le deja vivir. Al ser una terapia a base de un dispositivo tecnológico, cosa que hoy día está en auge, se siente como algo más ameno que de una terapia literal se tratara.

Un punto fuerte que destacar en estas conclusiones es el esfuerzo y la dedicación invertida en la realización de este proyecto que han conseguido, junto con la constancia, convertir la idea inicial propuesta para este trabajo en una aplicación totalmente funcional y listo para usar a pesar de todas las dificultades encontradas durante su realización debido al, hasta entonces, desconocimientos de las tecnologías implementadas y utilizadas.

Es por esto último, que puedo garantizar como autor del proyecto que gracias a su desarrollo he podido aprender multitud de conceptos, tecnologías y herramientas que hasta ahora desconocía y que, sin duda, enriquecen mi experiencia académica, habiéndome dado la oportunidad de ampliar mis conocimientos en el campo del desarrollo software y hardware, y personal al tener que hacer frente a todos los desafíos que se me han ido presentando durante el camino y habiendo descubierto una rama de la Ingeniería de Telecomunicaciones que me apasiona.

Finalmente, no puedo concluir este proyecto sin dar las gracias a mi tutora María del Mar Elena Pérez por proponerme este desafío y por ayudarme y asesorarme en todo momento.

# 9 REFERENCIAS


[1] 360cities.net. (2019). *Stock 360° Panoramic Images and Videos for VR and More | 360Cities*. [online] Available at: https://www.360cities.net [Accessed 28 Jul. 2019].

[2] Centrumpsicologos.com. (2019). *¿Qué es el EMDR es y Cómo Funciona? | Centrum Psicólogos*. [online] Available at: https://www.centrumpsicologos.com/terapia-emdr-funciona/ [Accessed 26 Aug. 2019].

[3] Facilitadores de Open Hardware. (2019). *Raspberry Pi – ejecución de programas en el arranque del S.O.*. [online] Available at: https://openhardware.pe/raspberry-pi-ejecucion-de-programas-en-el-arranque-del-s-o/ [Accessed 10 Jul. 2019].

[4] Flaticon. (2019). *Flaticon, la mayor base de datos de iconos vectoriales gratis*. [online] Available at: https://www.flaticon.es [Accessed 3 Sep. 2019].

[5] infootec.net. (2019). *Arduino Uno R3, tutorial especificaciones electrónicas y programación.*. [online] Available at: https://www.infootec.net/arduino/ [Accessed 22 Aug. 2019].

[6] Lucidchart.com. (2019). [online] Available at: https://www.lucidchart.com/pages/es?noHomepageRedirect=true [Accessed 8 Jul. 2019].

[7] Naylampmechatronics.com. (2019). *Tutorial MPU6050, Acelerómetro y Giroscopio*. [online] Available at: https://naylampmechatronics.com/blog/45_Tutorial-MPU6050-Acelerómetro-y-Giroscopio.html [Accessed 12 Jul. 2019].

[8] Psiconetwork. (2019). *QUÉ ES Y CÓMO FUNCIONA LA TERAPIA EMDR*. [online] Available at: https://psiconetwork.com/que-es-y-como-funciona-la-terapia-emdr/ [Accessed 26 Aug. 2019].

[9] YouTube. (2019). *ARDUINO & MPU-6050 | J_RPM*. [online] Available at: https://www.youtube.com/watch?v=qAqBTACXFPs&lc=z22ht1nhbm3ewvudcacdp431kwf4iu2xreebrd3ctdpw03c010c.1564377771570752&feature=em-comments [Accessed 4 Jul. 2019].

[10] YouTube. (2019). *Eye tracking with a webcam, matlab and arduino*. [online] Available at: https://www.youtube.com/watch?v=ztaJDjMQH68 [Accessed 9 Aug. 2019].

[11] YouTube. (2019). *Raspberry Pi - Expanding RAM with USB External Swap File*. [online] Available at: https://www.youtube.com/watch?v=tVfMNI3w6Dw [Accessed 17 Jul. 2019].

[12] Raspberry para novatos. (2019). *Raspberry para novatos - Aprende todo lo que no sepas de Raspberry*. [online] Available at: https://raspberryparanovatos.com [Accessed 9 Jul. 2019].

[13] GitHub. (2019). *IvanHalen86/Webcam EyeTracker Matlab Arduino*. [online] Available at: https://github.com/IvanHalen86/Webcam_EyeTracker_Matlab_Arduino [Accessed 15 Aug. 2019].

[14] Processing.org. (2019). *Processing.org*. [online] Available at: https://processing.org [Accessed 22 Jul. 2019].

# 10 ANEXO A: CÓDIGOS DEL PROYECTO

En el presente anexo se documentará todos y cada uno de los códigos explícitamente por si pueda llegar a suscitar a alguna duda la explicación de estos.

## 10.1 MPU_calibration.ino

```
1. // Arduino sketch that returns calibration offsets for MPU6050
2. // These offsets were meant to calibrate MPU6050's internal DMP, but
   can be also useful for reading sensors.
3. // The effect of temperature has not been taken into account so I can't
   promise that it will work if you
4. // calibrate indoors and then use it outdoors. Best is to calibrate and
   use at the same room temperature.
5.
6. // I2Cdev and MPU6050 must be installed as libraries
7. #include "I2Cdev.h"
8. #include "MPU6050.h"
9. #include "Wire.h"
10.
11.      ///////////////////////////////////   CONFIGURATION   ////////////
   /////////////////
12.      //Change this 3 variables if you want to fine tune the skecth to
   your needs.
13.      int buffersize=1000;     //Amount of readings used to average,
   make it higher to get more precision but sketch will be
   slower  (default:1000)
14.      int acel_deadzone=8;     //Acelerometer error allowed, make it
   lower to get more precision, but sketch may not converge  (default:8)
15.      int giro_deadzone=1;     //Giro error allowed, make it lower to
   get more precision, but sketch may not converge  (default:1)
16.
17.      // default I2C address is 0x68
18.      // specific I2C addresses may be passed as a parameter here
19.      // AD0 low = 0x68 (default for InvenSense evaluation board)
20.      // AD0 high = 0x69
21.      //MPU6050 accelgyro;
22.      MPU6050 accelgyro(0x68); // <-- use for AD0 high
23.
24.      int16_t ax, ay, az,gx, gy, gz;
25.
26.      int mean_ax,mean_ay,mean_az,mean_gx,mean_gy,mean_gz,state=0;
27.      int ax_offset,ay_offset,az_offset,gx_offset,gy_offset,gz_offset;
28.
29.      ///////////////////////////////////   SETUP   ////////////////////
   ////////////////
30.      void setup() {
31.        // join I2C bus (I2Cdev library doesn't do this automatically)
32.        Wire.begin();
33.        // COMMENT NEXT LINE IF YOU ARE USING ARDUINO DUE
34.        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Leonardo
   measured 250kHz.
```

```
35.
36.        // initialize serial communication
37.        Serial.begin(115200);
38.
39.        // initialize device
40.        accelgyro.initialize();
41.
42.        // wait for ready
43.        while (Serial.available() && Serial.read()); // empty buffer
44.        while (!Serial.available()){
45.          Serial.println(F("Send any character to start sketch.\n"));
46.          delay(1500);
47.        }
48.        while (Serial.available() && Serial.read()); // empty buffer
   again
49.
50.        // start message
51.        Serial.println("\nMPU6050 Calibration Sketch");
52.        delay(2000);
53.        Serial.println("\nYour MPU6050 should be placed in horizontal
   position, with package letters facing up. \nDon't touch it until you
   see a finish message.\n");
54.        delay(3000);
55.        // verify connection
56.        Serial.println(accelgyro.testConnection() ? "MPU6050 connection
   successful" : "MPU6050 connection failed");
57.        delay(1000);
58.        // reset offsets
59.        accelgyro.setXAccelOffset(0);
60.        accelgyro.setYAccelOffset(0);
61.        accelgyro.setZAccelOffset(0);
62.        accelgyro.setXGyroOffset(0);
63.        accelgyro.setYGyroOffset(0);
64.        accelgyro.setZGyroOffset(0);
65.      }
66.
67.      ////////////////////////////////   LOOP   ////////////////////
   ///////////////
68.      void loop() {
69.        if (state==0){
70.          Serial.println("\nReading sensors for first time...");
71.          meansensors();
72.          state++;
73.          delay(1000);
74.        }
75.
76.        if (state==1) {
77.          Serial.println("\nCalculating offsets...");
78.          calibration();
79.          state++;
80.          delay(1000);
81.        }
82.
83.        if (state==2) {
84.          meansensors();
85.          Serial.println("\nFINISHED!");
86.          Serial.print("\nSensor readings with offsets:\t");
87.          Serial.print(mean_ax);
88.          Serial.print("\t");
89.          Serial.print(mean_ay);
90.          Serial.print("\t");
91.          Serial.print(mean_az);
92.          Serial.print("\t");
93.          Serial.print(mean_gx);
```

```
94.           Serial.print("\t");
95.           Serial.print(mean_gy);
96.           Serial.print("\t");
97.           Serial.println(mean_gz);
98.           Serial.print("Your offsets:\t");
99.           Serial.print(ax_offset);
100.          Serial.print("\t");
101.          Serial.print(ay_offset);
102.          Serial.print("\t");
103.          Serial.print(az_offset);
104.          Serial.print("\t");
105.          Serial.print(gx_offset);
106.          Serial.print("\t");
107.          Serial.print(gy_offset);
108.          Serial.print("\t");
109.          Serial.println(gz_offset);
110.          Serial.println("\nData is printed as: acelX acelY acelZ giroX
   giroY giroZ");
111.          Serial.println("Check that your sensor readings are close to 0
   0 16384 0 0 0");
112.          Serial.println("If calibration was succesful write down your
   offsets so you can set them in your projects using something similar to
   mpu.setXAccelOffset(youroffset)");
113.          while (1);
114.        }
115.      }
116.
117.      ///////////////////////////////////   FUNCTIONS   ////////////////
   ///////////////////
118.      void meansensors(){
119.        long i=0,buff_ax=0,buff_ay=0,buff_az=0,buff_gx=0,buff_gy=0,buff_
   gz=0;
120.
121.        while (i<(buffersize+101)){
122.          // read raw accel/gyro measurements from device
123.          accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
124.
125.          if (i>100 && i<=(buffersize+100)){ //First 100 measures are
   discarded
126.            buff_ax=buff_ax+ax;
127.            buff_ay=buff_ay+ay;
128.            buff_az=buff_az+az;
129.            buff_gx=buff_gx+gx;
130.            buff_gy=buff_gy+gy;
131.            buff_gz=buff_gz+gz;
132.          }
133.          if (i==(buffersize+100)){
134.            mean_ax=buff_ax/buffersize;
135.            mean_ay=buff_ay/buffersize;
136.            mean_az=buff_az/buffersize;
137.            mean_gx=buff_gx/buffersize;
138.            mean_gy=buff_gy/buffersize;
139.            mean_gz=buff_gz/buffersize;
140.          }
141.          i++;
142.          delay(2); //Needed so we don't get repeated measures
143.        }
144.      }
145.
146.      void calibration(){
147.        ax_offset=-mean_ax/8;
148.        ay_offset=-mean_ay/8;
149.        az_offset=(16384-mean_az)/8;
150.
```

```
151.       gx_offset=-mean_gx/4;
152.       gy_offset=-mean_gy/4;
153.       gz_offset=-mean_gz/4;
154.       while (1){
155.         int ready=0;
156.         accelgyro.setXAccelOffset(ax_offset);
157.         accelgyro.setYAccelOffset(ay_offset);
158.         accelgyro.setZAccelOffset(az_offset);
159.
160.         accelgyro.setXGyroOffset(gx_offset);
161.         accelgyro.setYGyroOffset(gy_offset);
162.         accelgyro.setZGyroOffset(gz_offset);
163.
164.         meansensors();
165.         Serial.println("...");
166.
167.         if (abs(mean_ax)<=acel_deadzone) ready++;
168.         else ax_offset=ax_offset-mean_ax/acel_deadzone;
169.
170.         if (abs(mean_ay)<=acel_deadzone) ready++;
171.         else ay_offset=ay_offset-mean_ay/acel_deadzone;
172.
173.         if (abs(16384-mean_az)<=acel_deadzone) ready++;
174.         else az_offset=az_offset+(16384-mean_az)/acel_deadzone;
175.
176.         if (abs(mean_gx)<=giro_deadzone) ready++;
177.         else gx_offset=gx_offset-mean_gx/(giro_deadzone+1);
178.
179.         if (abs(mean_gy)<=giro_deadzone) ready++;
180.         else gy_offset=gy_offset-mean_gy/(giro_deadzone+1);
181.
182.         if (abs(mean_gz)<=giro_deadzone) ready++;
183.         else gz_offset=gz_offset-mean_gz/(giro_deadzone+1);
184.
185.         if (ready==6) break;
186.       }
187.     }
```

## 10.2 MPU_Arduino.ino

```
1.
2. // I2Cdev and MPU6050 must be installed as libraries, or else the
   .cpp/.h files
3. // for both classes must be in the include path of your project
4. #include "I2Cdev.h"
5. #include "MPU6050_6Axis_MotionApps20.h"
6.
7. //#include "MPU6050.h" // not necessary if using MotionApps include
   file
8.
9. // Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
   implementation
10.      // is used in I2Cdev.h
11.      #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
12.          #include "Wire.h"
13.      #endif
14.
15.      // class default I2C address is 0x68
16.      // specific I2C addresses may be passed as a parameter here
17.      // AD0 low = 0x68 (default for SparkFun breakout and InvenSense
   evaluation board)
18.      // AD0 high = 0x69
19.      MPU6050 mpu;
20.      //MPU6050 mpu(0x69); // <-- use for AD0 high
21.
22.      /*
   ========================================================================
   ==
23.         MPU6050 - ARDUINO UNO
24.           SDA   -   A4
25.           SCL   -   A5
26.           AD0   -   GND (AD0 low = 0x68 / AD0 high = 0x69)
27.           INT   -   PIN2
28.           VCC   -   3.3V
29.           GND   -   GND
30.       *
   ========================================================================
   == */
31.
32.
33.
34.      // uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the
   actual
35.      // quaternion components in a [w, x, y, z] format (not best for
   parsing
36.      // on a remote host such as Processing or something though)
37.      #define OUTPUT_READABLE_QUATERNION
38.
39.
40.      #define LED_PIN 13 // (LED Arduino en PIN 13)
41.      bool blinkState = false;
42.
43.      // MPU control/status vars
44.      bool dmpReady = false;  // set true if DMP init was successful
45.      uint8_t mpuIntStatus;   // holds actual interrupt status byte from
   MPU
46.      uint8_t devStatus;      // return status after each device
   operation (0 = success, !0 = error)
47.      uint16_t packetSize;    // expected DMP packet size (default is 42
   bytes)
```

```
48.      uint16_t fifoCount;     // count of all bytes currently in FIFO
49.      uint8_t fifoBuffer[64]; // FIFO storage buffer
50.
51.      // orientation/motion vars
52.      Quaternion q;           // [w, x, y, z]         quaternion
   container
53.
54.      //
   ============================================================
55.      // ===               INTERRUPT DETECTION
   ROUTINE                ===
56.      //
   ============================================================
57.
58.      volatile bool mpuInterrupt = false;     // indicates whether MPU
   interrupt pin has gone high
59.      void dmpDataReady() {
60.          mpuInterrupt = true;
61.      }
62.
63.      //
   ============================================================
64.      // ===                         INITIAL
   SETUP                       ===
65.      //
   ============================================================
66.
67.      void setup() {
68.          // join I2C bus (I2Cdev library doesn't do this automatically)
69.          #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
70.              Wire.begin();
71.              TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
72.          #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
73.              Fastwire::setup(400, true);
74.          #endif
75.
76.          // initialize serial communication
77.          // (115200 chosen because it is required for Teapot Demo
   output, but it's
78.          // really up to you depending on your project)
79.          Serial.begin(115200);
80.          while (!Serial); // wait for Leonardo enumeration, others
   continue immediately
81.
82.
83.          // initialize device
84.
85.          //Serial.println(F("Initializing I2C devices..."));
86.          mpu.initialize();
87.
88.          // load and configure the DMP
89.          ///Serial.println(F("Initializing DMP..."));
90.          devStatus = mpu.dmpInitialize();
91.
92.
93.      //
   ============================================================
94.      // The current calibration data
95.      //
   ============================================================
96.
97.          // supply your own gyro offsets here, scaled for min
   sensitivity
98.          mpu.setXGyroOffset(14);
```

```
99.          mpu.setYGyroOffset(7);
100.         mpu.setZGyroOffset(16);
101.         mpu.setZAccelOffset(999); // 16391 factory default for my test
   chip
102.
103.         // make sure it worked (returns 0 if so)
104.         if (devStatus == 0) {
105.             // turn on the DMP, now that it's ready
106.
107.             ////Serial.println(F("Enabling DMP..."));
108.             mpu.setDMPEnabled(true);
109.
110.             // enable Arduino interrupt detection
111.             ////Serial.println(F("Enabling interrupt detection
   (Arduino external interrupt 0)..."));
112.             attachInterrupt(0, dmpDataReady, RISING);
113.             mpuIntStatus = mpu.getIntStatus();
114.
115.             // set our DMP Ready flag so the main loop() function
   knows it's okay to use it
116.             ///Serial.println(F("DMP ready! Waiting for first
   interrupt..."));
117.             dmpReady = true;
118.
119.             // get expected DMP packet size for later comparison
120.             packetSize = mpu.dmpGetFIFOPacketSize();
121.         } else {
122.             // ERROR!
123.             // 1 = initial memory load failed
124.             // 2 = DMP configuration updates failed
125.             // (if it's going to break, usually the code will be 1)
126.             Serial.print(F("DMP Initialization failed (code "));
127.             Serial.print(devStatus);
128.             Serial.println(F(")"));
129.         }
130.
131.         // configure LED for output
132.         pinMode(LED_PIN, OUTPUT);
133.     }
134.
135.     //
   ================================================================
136.     // ===                    MAIN PROGRAM
   LOOP                    ===
137.     //
   ================================================================
138.
139.     void loop() {
140.
141.     ////////////////////////////////////////////////////////////////
   ///
142.     // ===  Delay to adjust the speed of the PC with PROCESSING
   3   ===
143.     // ...if the processor in your PC is fast, you can eliminate this
   delay
144.     ////////////////////////////////////////////////////////////////
   ///
145.         delay(200);
146.         mpu.resetFIFO();
147.         //delay(50);
148.     ////////////////////////////////////////////////////////////////
   ///
149.
150.         // if programming failed, don't try to do anything
```

```
151.         if (!dmpReady) return;
152.
153.         // wait for MPU interrupt or extra packet(s) available
154.         while (!mpuInterrupt && fifoCount < packetSize) {
155.             // other program behavior stuff here
156.             // .
157.             // .
158.             // .
159.             // if you are really paranoid you can frequently test in
   between other
160.             // stuff to see if mpuInterrupt is true, and if so,
   "break;" from the
161.             // while() loop to immediately process the MPU data
162.             // .
163.             // .
164.             // .
165.         }
166.
167.
168.         // reset interrupt flag and get INT_STATUS byte
169.         mpuInterrupt = false;
170.         mpuIntStatus = mpu.getIntStatus();
171.
172.         // get current FIFO count
173.         fifoCount = mpu.getFIFOCount();
174.
175.         // check for overflow (this should never happen unless our
   code is too inefficient)
176.         if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
177.             // reset so we can continue cleanly
178.             mpu.resetFIFO();
179.             Serial.println(F("FIFO overflow!"));
180.
181.         // otherwise, check for DMP data ready interrupt (this should
   happen frequently)
182.         } else if (mpuIntStatus & 0x02) {
183.             // wait for correct available data length, should be a
   VERY short wait
184.             while (fifoCount < packetSize) fifoCount = mpu.getFIFOCoun
   t();
185.
186.             // read a packet from FIFO
187.             mpu.getFIFOBytes(fifoBuffer, packetSize);
188.
189.             // track FIFO count here in case there is > 1 packet
   available
190.             // (this lets us immediately read more without waiting for
   an interrupt)
191.             fifoCount -= packetSize;
192.
193.             // New protocol for sending data [J_RPM]
194.             #ifdef OUTPUT_READABLE_QUATERNION
195.                 // display quaternion values in easy matrix form: w x
   y z
196.                 mpu.dmpGetQuaternion(&q, fifoBuffer);
197.                   Serial.print(q.w);
198.                   Serial.print(",");
199.                   Serial.print(q.x);
200.                   Serial.print(",");
201.                   Serial.print(q.y);
202.                   Serial.print(",");
203.                   Serial.print(q.z);
204.                   Serial.println(",");
205.             #endif
```

```
206.
207.
208.             // blink LED to indicate activity
209.             blinkState = !blinkState;
210.             digitalWrite(LED_PIN, blinkState);
211.
212.
213.
214.         }
215.     }
```

## 10.3 MPU_Processing.pde

```
1.
2. import processing.serial.*;
3.
4. Serial myPort;  // Create object from Serial class
5.
6.
7. final String serialPort = "/dev/ttyACM0";
8.
9.
10.      float [] q = new float [4];
11.      float [] hq = null;
12.      float [] Euler = new float [3]; // psi, theta, phi
13.
14.      int lf = 10; // 10 is '\n' in ASCII
15.      byte[] inBuffer = new byte[22]; // this is the number of chars on
   each line from the Arduino (including /r/n)
16.
17.      PFont font;
18.      final int VIEW_SIZE_X = 800, VIEW_SIZE_Y = 600;
19.
20.      //For load images PNG
21.      PImage
   anchoP,anchoTP,superiorP,inferiorP,estrechoIP,estrechoDP;//For logo
22.      PImage
   anchoPNG,anchoTPNG,superiorPNG,inferiorPNG,estrechoIPNG,estrechoDPNG;
23.      PImage
   anchoPNG2,anchoTPNG2,superiorPNG2,inferiorPNG2,estrechoIPNG2,estrechoDP
   NG2;
24.      PImage
   anchoPNG3,anchoTPNG3,superiorPNG3,inferiorPNG3,estrechoIPNG3,estrechoDP
   NG3;
25.      PImage fondo;
26.
27.      //For Splash Screen
28.      PImage splash;
29.
30.      //For load Gifs
31.      PImage[] ancho=new PImage[50];
32.      PImage[] superior=new PImage[50];
33.      PImage[] inferior=new PImage[50];
34.      PImage[] estrechoI=new PImage[50];
35.      PImage[] estrechoD=new PImage[50];
36.
37.      //For button
38.      int rectX, rectY;
39.      int rectSize=65;
40.      int circleX, circleY;
41.      int circleSize = 75;
42.      int triX, triY;
43.      int triSize=70;
44.      color rectColor, circleColor, triColor;
45.      color rectShadow, circleShadow, triShadow;
46.      boolean rectOver=false;
47.      boolean circleOver=false;
48.      boolean triOver=false;
49.
50.      //For change images
51.      int valueR=1;
52.      int valueC=0;
53.      int valueT=0;
```

```
54.      int state=0;
55.
56.
57.      void setup()
58.      {
59.        size(1200, 720, P3D);
60.        textureMode(NORMAL);
61.        fill(255);
62.        stroke(color(44,48,32));
63.        //noStroke();
64.
65.        myPort = new Serial(this, serialPort, 115200);
66.
67.        // The font must be located in the sketch's "data" directory to
   load successfully
68.        font = loadFont("Lato-Light-48.vlw");
69.
70.        //Button
71.        rectColor=color(255,0,0,255);
72.        rectShadow=color(200,0,0,255);
73.        circleColor = color(0,255,0,255);
74.        circleShadow= color(0,200,0,255);
75.        triColor=color(0);
76.        triShadow=color(51);
77.        rectX=675;
78.        rectY=630;
79.        circleX=505;
80.        circleY=660;
81.        triX=600;
82.        triY=650;
83.
84.        // Loading the textures to the cube
85.        // The png files alow to put the board holes so can
   increase  realism
86.        smooth();
87.
88.       //For Splash Screen
89.        splash=loadImage("splash1200.png");
90.        size(1200,720);
91.
92.       //For background
93.        fondo=loadImage("stars.png");
94.
95.       //Top Side PNG
96.        superiorPNG= loadImage("prof.png");
97.        superiorPNG2= loadImage("sup.png");
98.        superiorP= loadImage("EMDR.png");
99.        superiorPNG3= loadImage("superior3.png");
100.
101.
102.       //Botm side PNG
103.       inferiorPNG = loadImage("prof.png");
104.       inferiorPNG2= loadImage("inf.png");
105.       inferiorP= loadImage("EMDR.png");
106.       inferiorPNG3= loadImage("inferior3.png");
107.
108.
109.
110.       //Wide side PNG
111.       anchoPNG = loadImage("prof.png");
112.       anchoTPNG = loadImage("prof.png");
113.       anchoPNG2= loadImage("ancho2.png");
114.       anchoP= loadImage("EMDR.png");
115.       anchoTPNG2= loadImage("atras.png");
```

```
116.       anchoTP= loadImage("EMDR.png");
117.       anchoPNG3= loadImage("ancho3.png");
118.       anchoTPNG3= loadImage("anchoT3.png");
119.
120.
121.       // Narrow left side PNG
122.       estrechoIPNG = loadImage("prof.png");
123.       estrechoIPNG2= loadImage("izq2.png");
124.       estrechoIP= loadImage("EMDR.png");
125.       estrechoIPNG3= loadImage("izq3.png");
126.
127.
128.
129.       // Narrow rigth side PNG
130.       estrechoDPNG= loadImage("prof.png");
131.       estrechoDPNG2= loadImage("dcha2.png");
132.       estrechoDP= loadImage("EMDR.png");
133.       estrechoDPNG3= loadImage("dcha3.png");
134.
135.
136.
137.       delay(100);
138.       myPort.clear();
139.     }
140.
141.
142.     void readQ() {
143.       if(myPort.available() >= 20) {
144.         String inputString = myPort.readStringUntil('\n');
145.         print(inputString);
146.         if (inputString != null && inputString.length() > 0) {
147.           String [] inputStringArr = split(inputString, ",");
148.           if(inputStringArr.length >= 5) { // q1,q2,q3,q4,\r\n so we
   have 5 elements
149.
150.             // New conversion to float [J_RPM]
151.             q[0] = float(inputStringArr[0]);
152.             q[1] = float(inputStringArr[1]);
153.             q[2] = float(inputStringArr[2]);
154.             q[3] = float(inputStringArr[3]);
155.
156.           }
157.         }
158.       }
159.     }
160.
161.
162.     void superiorImg(PImage imag) {
163.       beginShape(QUADS);
164.
165.        if(valueT!=0){
166.         texture(imag);
167.        }
168.
169.       // -Y "top" face
170.       vertex(-1, -1, -1, 0, 0);
171.       vertex( 1, -1, -1, 1, 0);
172.       vertex( 1, -1,  1, 1, 1);
173.       vertex(-1, -1,  1, 0, 1);
174.
175.       endShape();
176.     }
177.
178.     void inferiorImg(PImage imag) {
```

```
179.       beginShape(QUADS);
180.
181.        if(valueT!=0){
182.         texture(imag);
183.        }
184.
185.
186.       // +Y "bottom" face
187.       vertex(-1,  1,  1, 0, 0);
188.       vertex( 1,  1,  1, 1, 0);
189.       vertex( 1,  1, -1, 1, 1);
190.       vertex(-1,  1, -1, 0, 1);
191.
192.       endShape();
193.     }
194.
195.
196.     void anchoImg(PImage imag) {
197.       beginShape(QUADS);
198.
199.        if(valueT!=0){
200.         texture(imag);
201.        }
202.
203.
204.       // +Z "front" face
205.       vertex(-1, -1,  1, 0, 0);
206.       vertex( 1, -1,  1, 1, 0);
207.       vertex( 1,  1,  1, 1, 1);
208.       vertex(-1,  1,  1, 0, 1);
209.
210.       endShape();
211.     }
212.
213.     void anchoTImg(PImage imag) {
214.       beginShape(QUADS);
215.
216.        if(valueT!=0){
217.         texture(imag);
218.        }
219.
220.
221.       // -Z "back" face
222.       vertex( 1, -1, -1, 0, 0);
223.       vertex(-1, -1, -1, 1, 0);
224.       vertex(-1,  1, -1, 1, 1);
225.       vertex( 1,  1, -1, 0, 1);
226.
227.
228.       endShape();
229.     }
230.
231.     void estrechoI(PImage imag) {
232.       beginShape(QUADS);
233.
234.        if(valueT!=0){
235.         texture(imag);
236.        }
237.
238.        // +X "right" face
239.       vertex( 1, -1,  1, 0, 0);
240.       vertex( 1, -1, -1, 1, 0);
241.       vertex( 1,  1, -1, 1, 1);
242.       vertex( 1,  1,  1, 0, 1);
```

```
243.
244.       endShape();
245.     }
246.
247.
248.     void estrechoD(PImage imag) {
249.       beginShape(QUADS);
250.        if(valueT!=0){
251.         texture(imag);
252.        }
253.
254.       // -X "left" face
255.       vertex(-1, -1, -1, 0, 0);
256.       vertex(-1, -1,  1, 1, 0);
257.       vertex(-1,  1,  1, 1, 1);
258.       vertex(-1,  1, -1, 0, 1);
259.
260.       endShape();
261.     }
262.
263.     void drawCube() {
264.       pushMatrix();
265.         translate(VIEW_SIZE_X-480, VIEW_SIZE_Y-270 , 0);
266.         scale(130);
267.         fill(0,0,255);
268.
269.         rotateZ(-Euler[2]);
270.         rotateX(-Euler[1]);
271.         rotateY(-Euler[0]);
272.
273.
274.           superiorImg(superiorP);
275.           inferiorImg(inferiorP);
276.           anchoImg(anchoP);
277.           anchoTImg(anchoTP);
278.           estrechoI(estrechoIP);
279.           estrechoD(estrechoDP);
280.
281.          if(valueT==1){
282.
283.           superiorImg(superiorPNG3);
284.           inferiorImg(inferiorPNG3);
285.           anchoImg(anchoPNG3);
286.           anchoTImg(anchoTPNG3);
287.           estrechoI(estrechoIPNG3);
288.           estrechoD(estrechoDPNG3);
289.         }
290.
291.           else if(valueT==2){
292.           superiorImg(superiorPNG2);
293.           inferiorImg(inferiorPNG2);
294.           anchoImg(anchoPNG2);
295.           anchoTImg(anchoTPNG2);
296.           estrechoI(estrechoIPNG2);
297.           estrechoD(estrechoDPNG2);
298.         }
299.
300.         else{
301.         valueT=0;
302.         }
303.
304.       popMatrix();
305.
306.
```

```
307.       pushMatrix();
308.         translate(VIEW_SIZE_X+100, VIEW_SIZE_Y-270 , 0);
309.         scale(130);
310.         fill(255,0,0);
311.
312.         rotateZ(-Euler[2]);
313.         rotateX(-Euler[1]);
314.         rotateY(-Euler[0]);
315.
316.
317.           superiorImg(superiorP);
318.           inferiorImg(inferiorP);
319.           anchoImg(anchoP);
320.           anchoTImg(anchoTP);
321.           estrechoI(estrechoIP);
322.           estrechoD(estrechoDP);
323.
324.          if(valueT==1){
325.
326.           superiorImg(superiorPNG3);
327.           inferiorImg(inferiorPNG3);
328.           anchoImg(anchoPNG3);
329.           anchoTImg(anchoTPNG3);
330.           estrechoI(estrechoIPNG3);
331.           estrechoD(estrechoDPNG3);
332.         }
333.
334.           else if(valueT==2){
335.           superiorImg(superiorPNG2);
336.           inferiorImg(inferiorPNG2);
337.           anchoImg(anchoPNG2);
338.           anchoTImg(anchoTPNG2);
339.           estrechoI(estrechoIPNG2);
340.           estrechoD(estrechoDPNG2);
341.         }
342.
343.         else{
344.         valueT=0;
345.         }
346.
347.       popMatrix();
348.
349.     }
350.
351.     void draw() {
352.
353.       if(state==0){
354.
355.         image(splash,0,0);
356.         if(millis()>=12000){
357.          state=1;
358.         }
359.       }
360.
361.      else{
362.
363.      if(valueC==0 && valueR==1){
364.       valueT=0;
365.       background(#140d4d);
366.       background(fondo);
367.       strokeWeight(3);
368.       readQ();
369.
370.       if(hq != null) { // use home quaternion
```

```
371.         quaternionToEuler(quatProd(hq, q), Euler);
372.       }
373.       else {
374.         quaternionToEuler(q, Euler);
375.       }
376.
377.       //Draw button
378.       update(mouseX, mouseY);
379.
380.       if(rectOver){
381.         fill(rectShadow);
382.       }
383.       else{
384.         fill(rectColor);
385.       }
386.
387.       stroke(255);
388.       rect(rectX,rectY,rectSize,rectSize);
389.
390.       if (circleOver) {
391.         fill(circleShadow);
392.       } else {
393.         fill(circleColor);
394.       }
395.       stroke(255);
396.       ellipse(circleX, circleY, circleSize, circleSize);
397.
398.       if(triOver){
399.         fill(triShadow);
400.       }
401.       else{
402.         fill(triColor);
403.       }
404.
405.       stroke(255);
406.
407.       triangle(580,700,580,620,645,660);
408.
409.      }
410.
411.      else{
412.       background(#140d4d);
413.       background(fondo);
414.       readQ();
415.
416.       if(hq != null) { // use home quaternion
417.         quaternionToEuler(quatProd(hq, q), Euler);
418.
419.       }
420.       else {
421.         quaternionToEuler(q, Euler);
422.
423.       }
424.
425.       textFont(font, 30);
426.       fill(255,0,255);
427.       textAlign(LEFT, TOP);
428.       text("Virtual Reality\n",20,20);
429.
430.
431.       strokeWeight(0.03);
432.       stroke(0);
433.       drawCube();
434.       strokeWeight(3);
```

```
435.       stroke(255);
436.
437.
438.       stroke(255);
439.       //Draw button
440.       update(mouseX, mouseY);
441.
442.       if(rectOver){
443.         fill(rectShadow);
444.       }
445.       else{
446.         fill(rectColor);
447.       }
448.
449.       stroke(255);
450.       rect(rectX,rectY,rectSize,rectSize);
451.
452.       if (circleOver) {
453.         fill(circleShadow);
454.       } else {
455.         fill(circleColor);
456.       }
457.       stroke(255);
458.       ellipse(circleX, circleY, circleSize, circleSize);
459.
460.       if(triOver){
461.         fill(triShadow);
462.       }
463.       else{
464.         fill(triColor);
465.       }
466.
467.       stroke(255);
468.       triangle(580,700,580,620,645,660);
469.       }
470.      }
471.     }
472.
473.
474.     void update(int x, int y){
475.       if ( overCircle(circleX, circleY, circleSize) ) {
476.         circleOver = true;
477.         rectOver = triOver= false;
478.       } else if ( overRect(rectX, rectY, rectSize, rectSize) ) {
479.         rectOver = true;
480.         circleOver =triOver= false;
481.       } else if( overTri(triX, triY, triSize, triSize)){
482.         triOver = true;
483.         circleOver = rectOver = false;
484.
485.       } else {
486.         circleOver = rectOver = triOver = false;
487.       }
488.
489.     }
490.
491.     boolean overRect(int x, int y, int width, int height){
492.      if(mouseX >= x && mouseX <= x+width && mouseY >= y && mouseY <= y
   +width){
493.        return true;
494.      }
495.      else{
496.        return false;
497.      }
```

```
498.
499.     }
500.
501.
502.     boolean overTri(int x, int y, int width, int height){
503.      if(mouseX >= x && mouseX <= x+width && mouseY >= y && mouseY <= y
   +width){
504.        return true;
505.      }
506.      else{
507.        return false;
508.      }
509.
510.     }
511.
512.     boolean overCircle(int x, int y, int diameter) {
513.       float disX = x - mouseX;
514.       float disY = y - mouseY;
515.       if (sqrt(sq(disX) + sq(disY)) < diameter/2 ) {
516.         return true;
517.       } else {
518.         return false;
519.       }
520.     }
521.
522.     void mousePressed(){
523.
524.      //For triangle button
525.      if(triOver && valueT==0 && valueC==1 && valueR==0){
526.       valueT=1;
527.      }
528.
529.      else if(triOver && valueT==1 && valueC==1 && valueR==0){
530.       valueT=2;
531.      }
532.
533.      else if(triOver && valueT==2 && valueC==1 && valueR==0){
534.       valueT=0;
535.      }
536.
537.      //For sqaure button
538.      if(rectOver && valueC==1 && valueR==0){
539.       valueR=1;
540.       valueC=0;
541.      }
542.
543.
544.      //For circle button
545.      if(circleOver && valueC==0 && valueR==1){
546.       valueC=1;
547.       valueR=0;
548.      }
549.
550.     }
551.
552.     void keyPressed() {
553.       if(key == 'h') {
554.         println("pressed h");
555.
556.         // set hq the home quaternion as the quatnion conjugate coming
   from the sensor fusion
557.         hq = quatConjugate(q);
558.
559.       }
```

```
560.       else if(key == 'n') {
561.         println("pressed n");
562.         hq = null;
563.       }
564.     }
565.
566.     // See Sebastian O.H. Madwick report
567.     // "An efficient orientation filter for inertial and
   intertial/magnetic sensor arrays" Chapter 2 Quaternion representation
568.
569.     void quaternionToEuler(float [] q, float [] euler) {
570.       euler[0] = atan2(2 * q[1] * q[2] - 2 * q[0] * q[3], 2 * q[0]*q[0
   ] + 2 * q[1] * q[1] - 1); // psi
571.       euler[1] = -asin(2 * q[1] * q[3] + 2 * q[0] * q[2]); // theta
572.       euler[2] = atan2(2 * q[2] * q[3] - 2 * q[0] * q[1], 2 * q[0] * q
   [0] + 2 * q[3] * q[3] - 1); // phi
573.
574.
575.     // When Euler[1] fails, it get  the maximum value (theta) [J_RPM]
576.       String theta = nf(Euler[1],2,7);
577.       //println(theta);
578.       if(theta.length() < 8)  {
579.         Euler[1] = 1.5;
580.       }
581.     }
582.
583.     float [] quatProd(float [] a, float [] b) {
584.       float [] q = new float[4];
585.
586.       q[0] = a[0] * b[0] - a[1] * b[1] - a[2] * b[2] - a[3] * b[3];
587.       q[1] = a[0] * b[1] + a[1] * b[0] + a[2] * b[3] - a[3] * b[2];
588.       q[2] = a[0] * b[2] - a[1] * b[3] + a[2] * b[0] + a[3] * b[1];
589.       q[3] = a[0] * b[3] + a[1] * b[2] - a[2] * b[1] + a[3] * b[0];
590.
591.       return q;
592.     }
593.
594.     // returns a quaternion from an axis angle representation
595.     float [] quatAxisAngle(float [] axis, float angle) {
596.       float [] q = new float[4];
597.
598.       float halfAngle = angle / 2.0;
599.       float sinHalfAngle = sin(halfAngle);
600.       q[0] = cos(halfAngle);
601.       q[1] = -axis[0] * sinHalfAngle;
602.       q[2] = -axis[1] * sinHalfAngle;
603.       q[3] = -axis[2] * sinHalfAngle;
604.
605.       return q;
606.     }
607.
608.     // return the quaternion conjugate of quat
609.     float [] quatConjugate(float [] quat) {
610.       float [] conj = new float[4];
611.
612.       conj[0] = quat[0];
613.       conj[1] = -quat[1];
614.       conj[2] = -quat[2];
615.       conj[3] = -quat[3];
616.
617.       return conj;
618.
```

## 10.4 Ardunio_MATLAB_Serial.ino

```
#define LED_PIN 13
bool blinkState = false;

int ledPin2 = 2;


int matlabData;

void setup()
      {
        // configure LED for output
        pinMode(ledPin2,OUTPUT);
        pinMode(LED_PIN, OUTPUT);

        Serial.begin(9600);
      }

      void loop()
      {

         if(Serial.available()>0) // if there is data to read
         {
          matlabData=Serial.read(); // read data

           if (matlabData==3)
           //blinkState = !blinkState;
           digitalWrite(ledPin2,HIGH); // turn light on
        }

          blinkState = !blinkState;
          //digitalWrite(LED_PIN, blinkState);
          digitalWrite(LED_PIN,blinkState); // turn light on
      }
```

## 10.5 EyeTracking.m

```
clc
clear all

if ~isempty(instrfind)
    fclose(instrfind);
    delete(instrfind);
end

% create an arduino object. instead of "/dev/cu.usbmodem1411','BaudRate"
      % use your serail port name. You can check it in arduino -> tools -> port.
      % Write the name of your port before the command 'BaudRate' in the
      % following string.
      %ar = arduino('/dev/tty.usbmodem14101');
      ar = serial('/dev/cu.SLAB_USBtoUART','BaudRate',9600);
      % Check for the camera name with the command
      % webcamlist() in matlab.
      cameras= webcamlist()
      % type "cameras" in the command window and see which camera correspond to your modified webcam. select that camera
      % just by entering the index within the curly brackets.
      % It could be 1, 2, 3... depends on hwo many webcams you have acctached on your computer
      mycamera = cameras{2}
      % create a 'cam' variable with your camera name
      cam = webcam(char(mycamera))

      %open arduino USB port
      fopen(ar);

      % send the pulse to the port number 2 of arduino
      fprintf(ar,'%s',char(3));

      pause(5);

      %check if the camera is positioned in the right way, with the focus on the
      %center of the eye. The "preview" command let you see what the camera see.

      preview(cam)


      % preallocate the matrix with the cooridnates data
      coordinate = zeros(50,2);

      for i = 1:50;

      % take a snapshot
      img = snapshot(cam);

      % calculate the center of the image

      filas=size(img,1);
      columnas=size(img,2);
      % Center
      centro_fila=round(filas/2);
      centro_columna=round(columnas/2);
      % transform the image in a BW image
```

```
55.       if size(img,3)==3
56.           la_imagen=img;
57.       end
58.
59.        piel=~im2bw(la_imagen,0.1);
60.       %     --
61.        piel=bwmorph(piel,'close');
62.        piel=bwmorph(piel,'open');
63.        piel=bwareaopen(piel,200);
64.        piel=imfill(piel,'holes');
65.
66.       % Tagged objects in BW image
67.       L=bwlabel(piel);
68.       % Get areas and tracking rectangle
69.       out_a=regionprops(L);
70.       % Count the number of objects
71.       N=size(out_a,1);
72.       while N < 1 || isempty(out_a) % Returns if no object in the image
73.           solo_cara=[ ];
74.           continue
75.       end
76.
77.       % Select the area
78.       areas=[out_a.Area];
79.       [area_min pam_min]=min(areas);
80.       [area_max pam_max]=max(areas);
81.
82.       % since there is a problem with the shades in the BW images (the
   algorithm detect the size of the black
83.       % shades in th eimage), we have to create an if statement where we
   declare that if the algorithm
84.       % detect a too big black area (threshold set on 10000), it has to
   consider the
85.       % smallest detected area as the pupil.
86.       % On the other hand, if the black area is below a treshold of
87.       % 10000 it considers the biggest black area as the pupil.
88.
89.       if area_max > 20000;
90.
91.       % show the BW image
92.       imagesc(la_imagen);
93.       colormap gray
94.       hold on
95.       % draw the red area around the pupil
96.       rectangle('Position',out_a(pam_min).BoundingBox,'EdgeColor',[1 0 0
   ],...
97.           'Curvature', [1,1],'LineWidth',2)
98.       centro=round(out_a(pam_min).Centroid);
99.
100.      % detect the X and Y coordinates
101.      X=centro(1);
102.      Y=centro(2);
103.
104.      % save X and Y coordinates in the coordinates matrix
105.      coordinate(i,1) = X;
106.      coordinate(i,2) = Y;
107.
108.      % draw the cross at the center of the pupil
109.      plot(X,Y,'g+')
110.      %
111.      text(X+10,Y,['(',num2str(X),',',num2str(Y),')'],'Color',[1 1 1])
112.      if X<centro_columna && Y<centro_fila
113.          title('Top left')
114.      elseif X>centro_columna && Y<centro_fila
```

```
115.         title('Top right')
116.     elseif X<centro_columna && Y>centro_fila
117.         title('Bottom left')
118.     else
119.         title('Bottom right')
120.     end
121.
122.     elseif area_max < 20000;
123.
124.     % show the BW image
125.     imagesc(la_imagen);
126.     colormap gray
127.     hold on
128.     % draw the red area around the pupil
129.     rectangle('Position',out_a(pam_max).BoundingBox,'EdgeColor',[1 0 0
    ],...
130.         'Curvature', [1,1],'LineWidth',2)
131.     centro=round(out_a(pam_max).Centroid);
132.
133.     % detect the X and Y coordinates
134.     X=centro(1);
135.     Y=centro(2);
136.
137.     % save X and Y coordinates in the coordinates matrix
138.     coordinate(i,1) = X;
139.     coordinate(i,2) = Y;
140.
141.     % draw the cross at the center of the pupil
142.     plot(X,Y,'g+')
143.     %
144.     text(X+10,Y,['(',num2str(X),',',num2str(Y),')'],'Color',[1 1 1])
145.     if X<centro_columna && Y<centro_fila
146.         title('Top left')
147.     elseif X>centro_columna && Y<centro_fila
148.         title('Top right')
149.     elseif X<centro_columna && Y>centro_fila
150.         title('Bottom left')
151.     else
152.         title('Bottom right')
153.     end
154.
155.     end
156.
157.     %% save current figure in a folder. IMPORTANT !!! Set your
    directory in the following command
158.     saveas(gcf,['/Users/AdanMonteroTorres/MATLAB/TFG/Eye',sprintf('%03
    d', i),'.png']);
159.
160.     end
161.
162.     %close arduino port number 2
163.     fprintf(ar,'%s',char(4));
164.     % close arduino USB port
165.     fclose(ar);
```

# 11 ANEXO B: INSTALACIÓN RASPBIAN

Cómo ya hemos hablado, a nuestra Raspberry Pi, le hemos instalado el sistema operativo basado en Debian llamado Raspbian, y para ello hemos seguido los siguientes pasos.

Lo primero que deberemos hacer será descargar el programa **SD Card Formatter** para formatear la tarjeta micro sd, de 32GB en nuestro caso, en formato FAT32, y para ello hemos de seleccionar la opción "*Overwritte format*", una vez introducida nuestra tarjeta.

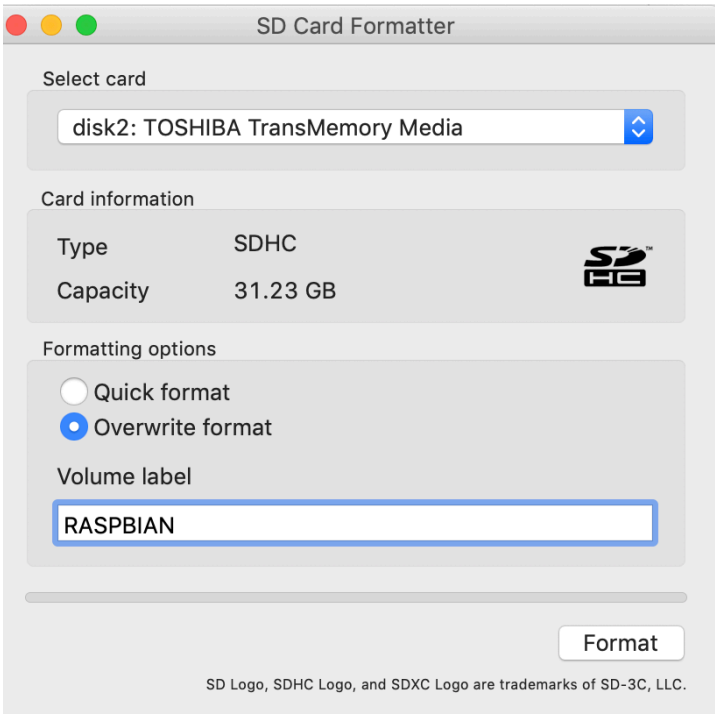



*Ilustración 58: Programa para formatear en FAT32*

Una vez formateado, nos dirigiremos a la página oficial de Raspberry para descargar el sistema operativo **Raspbian** y nos descargamos la opción recomendada.

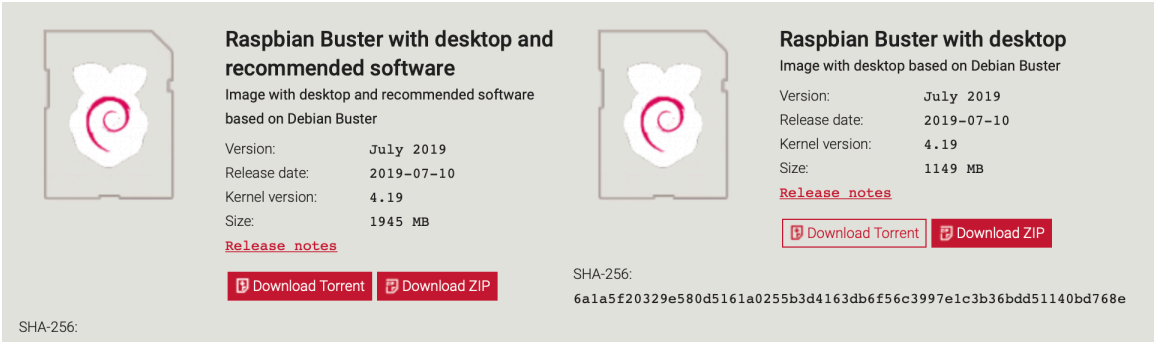



*Ilustración 59: Descarga de Raspbian*

Por último, nos descargamos el programa **balenaEtcher**, y seguimos los pasos indicados, es decir, seleccionamos la imagen descargada anteriormente, la sd formateada, pulsamos "Flash!" y cuando finalice, tendremos nuestra sd con el sistema operativo Raspbian listo para usar en nuestra Raspberry Pi.

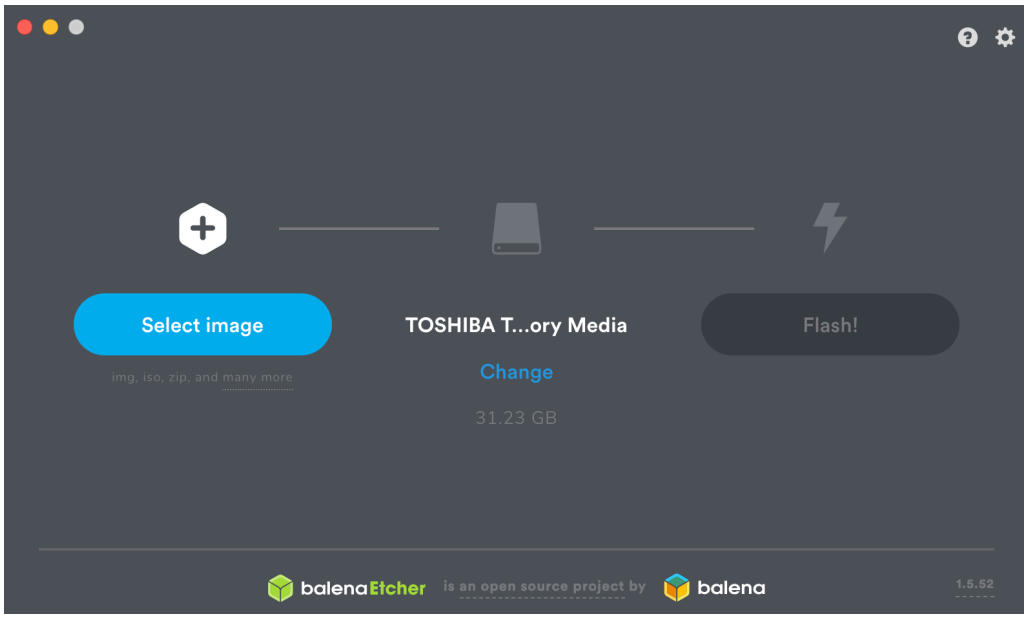



*Ilustración 60: Programa para grabar Raspbian*

# 12 ANEXO C: REQUISITOS ESPECIALES

Cómo antes hemos dicho anteriormente, este proyecto es una continuación de un trabajo final, por tanto, en dicho trabajo se hicieron unos requerimientos especiales como veremos a continuación.

## 12.1 Funcionalidades

F.1: El dispositivo debe ofrecer una experiencia de inmersión con gafas VR.

F.2: El dispositivo deberá realizar un rastreo del movimiento ocular.

F.3: El sistema debe estar compuesto de estímulos sonoros bilaterales. (o audio 3D)

F.4: El sistema debe tener estímulos táctiles con guantes sensoriales.

## 12.2 Prestaciones

P.1: Las gafas proyectarán un entorno virtual similar a un videojuego, donde el paciente se sentirá totalmente sumergido. Aparte, se programarán en dichos entornos avatares o ítems cercanos al paciente, de forma que la experiencia será más impactante y personal.

- Resolución de imagen: 2560x1440
- Fotogramas: 70fps.
- Conexiones: HDMI 1.4, USB 3.0

P.2: El sensor realizará el rastreo ocular cada 8,3ms (120fps).

P.3: El audio estará activado durante de toda la sesión.

## 12.3 Diseño

D.1: Las gafas deben estar adaptadas a los pacientes, ser ajustables, y deben ser ligeras.

- Peso: 500g
- Correas de velcro ajustables

D.2: Las gafas de realidad virtual han de incluir un sensor de rastreo ocular.

- Un sensor en cada lente.
- Precisión de rastreo.

D.3: Los altavoces deben ser estéreo de potencia X, con trípode regulable.

## 12.4 Operación

O.1: El sistema completo de Realidad Virtual debe estar conectado y configurado a través de un ordenador central.

O.2: Sería recomendable instalar el sistema completo en una sala con un bajo nivel de ruido externo.

## 12.5 Pruebas a realizar

Se realizarán pruebas a los distintos dispositivos para comprobar su correcto funcionamiento. Los dispositivos son los siguientes:

Sensor integrado en las gafas VR: la prueba consistirá en comprobar si el sensor realiza correctamente el seguimiento ocular del paciente. Un operario fijará la mirada en unos puntos predefinidos y sé comprobará la precisión del sensor en su seguimiento.

Altavoces: prueba de sonido para comprobar el carácter inmersivo referente al audio.

Al finalizar el desarrollo del producto se realizarán las siguientes pruebas:

- Se someterá el tracking de las gafas a un seguimiento exhaustivo del globo ocular, con movimientos completos del ojo con la ayuda de un voluntario.

- Se realizarán simulaciones en los diferentes entornos creados por el programa, comprobando su carácter inmersivo y realista.

- Se comprobará que el programa reproduce fielmente los entornos en función de las reacciones del paciente. Para ello un voluntario realizará todas las posibles combinaciones dentro de lo estipulado con tal de asegurar el correcto y completo funcionamiento del sistema.

Algunas de las pruebas que se realizarán serán con voluntarios específicos, a ser posible pacientes de la clínica encargada, con traumas. Éstas son las siguientes:

- Pruebas referentes al software. Se comprobará que el programa identifique correctamente el trauma que padece el paciente en cuestión. Estos voluntarios tendrán diagnosticado su respectivo trauma, y se comparará el veredicto final del software con el trauma ya declarado anteriormente.

Además, las pruebas anteriormente citadas serán cotejadas con el estudio plasmado en el libro “Desensibilización y reprocesamiento por medio del movimiento ocular” de la autora, neuróloga y psicoterapeuta cognitivo-comportamental, Francine Shapiro.

## 12.6 Normativa

Todo este proyecto cumplirá los requisitos recogidos en el Sistema de Gestión de Calidad de productos sanitarios redactada en la Norma ISO 13485:2013 (actualizada a 2016).

Esta norma indica que el producto puede ser utilizado por una organización, en el diseño y desarrollo, producción, instalación o prestación de servicios de productos sanitarios. Utilizar productos sanitarios que cumplan con esta norma, contribuye a aumentar la calidad de la asistencia sanitaria prestada.

Incorpora exigencias adicionales para poder dar cumplimiento a los requisitos legales del sector en aspectos como: diseño, control de los registros, procesos críticos, trazabilidad, procesos de esterilización, o gestión del riesgo.

Finalmente, lo que obtiene el cliente una vez superado el proceso de Auditoría, si el sistema implantado se adecúa a los requisitos de la norma mencionada, la organización obtiene:

- El certificado AENOR de Sistema de Gestión de la Calidad para Productos sanitarios.

- La licencia de uso de la marca de Sistema de Gestión de la Calidad para Productos sanitarios, de AENOR.

- El certificado IQNet, pasaporte para un acceso internacional de su certificación. Con él, su certificado AENOR quedará reconocido por las entidades de certificación líderes en el ámbito internacional.

- La licencia de uso de la marca IQNet.


*Ilustración 61: Normativa AENOR e IQNet*